13G3061/2USAAT

日本国特許庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: １９９９年　９月２２日

出願番号
Application Number: 平成１１年特許願第２６７８８９号

出願人
Applicant (s): 株式会社東芝

CERTIFIED COPY OF
PRIORITY DOCUMENT

２０００年　６月　９日

特許庁長官
Commissioner,
Patent Office 近藤隆彦

出証番号　出証特２０００－３０４４２４２

【書類名】　特許願

【整理番号】　13A9990261

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/38

【発明の名称】　中央演算装置、コンパイル方法、及びコンパイルプログラムを記録した記録媒体

【請求項の数】　11

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内

【氏名】　吉川　宜史

【特許出願人】

【識別番号】　000003078

【氏名又は名称】　株式会社　東芝

【代理人】

【識別番号】　100083161

【弁理士】

【氏名又は名称】　外川　英明

【電話番号】　03-3457-2512

【手数料の表示】

【予納台帳番号】　010261

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　中央演算装置、コンパイル方法、及びコンパイルプログラムを記録した記録媒体

【特許請求の範囲】

【請求項１】

複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置において、

前記プログラムが予め定められたデータ依存性を有する命令列の単位で区切られ、各命令列間の制御依存性は所定の条件により該命令列を承認もしくは否認するコミット命令により表現され、該命令列間のデータ依存性は該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与することにより表現され、

前記データ依存性を有する命令列が該命令列の単位で区切られたプログラム区間のいずれに属する命令列であるかを識別するための識別番号を該命令列に含まれる命令毎に付与する識別番号付与手段と、前記データ依存性を有する命令列の単位で命令を前記複数のバッファに割り当てる演算器割当手段と、特定の命令列が前記コミット命令の実行により承認された場合にのみレジスタを更新させるレジスタ更新手段と、特定の命令列が前記コミット命令の実行により承認された場合にのみメモリを更新させるメモリ更新手段とを備えたことを特徴とする中央演算装置。

【請求項２】

前記識別番号付与手段は、

直前の命令に付与した識別番号を保持する識別番号カウンタ手段と、

識別番号を付与すべき命令が前記コミット命令以外の命令である場合には、前記カウンタ手段に保持されている識別番号と同じ識別番号を付与し、識別番号を付与すべき命令が前記コミット命令である場合には、前記識別番号カウンタ手段に保持されている識別番号を１だけ増加させた識別番号を付与する手段とを含むことを特徴とする請求項１に記載の中央演算装置。

【請求項３】

前記演算器割当手段は、

前記データ依存性を有する命令列が、前記複数のバッファのいずれに割り当てられているかを示す情報を保持し、

ある命令が属する前記データ依存性を有する命令列に対するバッファの割り当てが存在する場合には、前記情報を参照して該命令を該命令列に対応するバッファに割り当て、該命令列に対するバッファの割り当てが存在しない場合には、前記複数のバッファの中から、処理が行なわれていないバッファを選んで該命令を割り当てることを特徴とする請求項１に記載の中央演算装置。

【請求項４】

前記演算器割当手段は、

予め定められた規則により各データ依存性を有する命令列を前記複数のバッファに割り当てることを特徴とする請求項１に記載の中央演算装置。

【請求項５】

前記レジスタ更新手段は、

前記複数のバッファから特定の命令が消去される際に、前記特定の命令を含む前記データ依存性を有する命令列が前記コミット命令の実行により承認された場合には、該特定の命令が特定のレジスタに退避した結果を用いてレジスタを更新し、該命令列が前記コミット命令の実行により否認された場合にはレジスタの更新を行なわないことを特徴とする請求項１に記載の中央演算装置。

【請求項６】

前記メモリ更新手段は、

前記演算実行ユニットで実行されたストア命令に関する情報を保持し、前記複数のバッファから前記ストア命令が消去される際に、前記ストア命令が含まれる前記データ依存性を有する命令列が前記コミット命令の実行により承認された場合には該ストア命令が登録した情報を用いてメモリを更新し、該命令列が前記コミット命令の実行により否認された場合にはメモリの更新を行なわないことを特徴とする請求項１に記載の中央演算装置。

【請求項７】

前記フラグが付与されたレジスタを使用する命令をデコードする際に、該レジ

スタを使用不可とし、該レジスタが命令により更新された際に該レジスタを使用可とすることを特徴とする請求項１に記載の中央演算装置。

【請求項８】

前記演算実行ユニットは、

特定の命令の実行結果を命令の出力オペランドとは異なるレジスタに一旦退避し、該特定の命令が含まれる前記データ依存性を有する命令列が前記コミット命令の実行により承認されたのちに、該レジスタの情報をレジスタ更新手段に送ることを特徴とする請求項１に記載の中央演算装置。

【請求項９】

前記複数のバッファから該演算実行ユニットに送ることが可能な命令が複数存在する場合に、投機的でない命令を投機的な命令より優先させて該演算実行ユニットに送ることを特徴とする請求項１に記載の中央演算装置。

【請求項１０】

複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置で実行されるプログラムを生成するコンパイル方法であって、

前記プログラムを予め定められたデータ依存性を有する命令列の単位で区切ったときに、該命令列のいずれに属する命令であるかを識別するための識別番号を命令毎に付与するとともに、該命令列間の制御依存性を表現するために、該命令列の承認もしくは否認を行なうコミット命令を発行し、該命令列間のデータ依存性を表現するために、該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与することを特徴とするコンパイル方法。

【請求項１１】

複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置で実行されるプログラムを生成するコンパイル手順であって、

前記プログラムを予め定められたデータ依存性を有する命令列の単位で区切ったときに、該命令列のいずれに属する命令であるかを識別するための識別番号を

命令毎に付与するとともに、該命令列間の制御依存性を表現するために、該命令列の承認もしくは否認を行なうコミット命令を発行し、該命令列間のデータ依存性を表現するために、該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与する手順を含むコンパイル手順をコンピュータで実行するためのコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、複数の命令を並列実行可能な中央演算装置、中央演算装置で実行されるプログラムを生成するコンパイル方法、およびコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体に関する。

【０００２】

【従来の技術】

命令並列度の向上は、計算機システムの高性能化を図る方法の一つである。制御依存性（条件分岐先の命令は、分岐条件を出力する演算が完了して条件が確定するまでは実行できない）や、データ依存性（命令は、その入力を生成する全ての命令が完了するまでは実行できない）は並列度を向上させる上での大きな障害となり、十分な命令並列度を得るためにはこれらの依存性を投機的に解除する方式が必要となる。

【０００３】

互いに依存性のある命令を並列実行する従来の方式としては、制御依存性についてはｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎと分岐予測があり、データ依存性についてはｄｅｐｅｎｄｅｎｃｅ　ｃｏｌｌａｐｓｉｎｇとｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎがある。以下、これらの方式について説明する。

【０００４】

ｐｒｅｄｉｃａｔｅｄ　ｅｘｅｕｔｉｏｎは命令にｐｒｅｄｉｃａｔｅと呼ばれる新たなソースオペランドを付加し、ｐｒｅｄｉｃａｔｅの真偽によりその命令が実行されるべきかどうかを表現するものである。

【０００５】

図２７にｐｒｅｄｉｃａｔｅ付き命令を含まないプログラムコードの一例を示す。この例では、条件分岐命令ｂｅｑにより、レジスタｒ３の値とレジスタｒ４の値とが等しい場合にはラベルＬ２に分岐し、等しくない場合にはｂｅｑの次の命令が実行される。

【０００６】

図２８は図２７のプログラムコードと同じ内容をｐｒｅｄｉｃａｔｅ用いて表したものである。（１）のｐｓｅｑ命令（ｐｒｅｄｉｃａｔｅセット命令）は、条件が成立した場合、本例ではレジスタｒ３の値とレジスタｒ４の値とが等しい場合に、ｐ１にｐｒｅｄｉｃａｔｅ値として１を、ｐ２にｐｒｅｄｉｃａｔｅ値として０をセットする。一方、条件が成立しない場合、本例ではレジスタｒ３の値とレジスタｒ４の値とが等しくない場合に、ｐ１を０に、ｐ２を１にセットする。

【０００７】

また、図２８の（２）～（４）のように、＜ｐ１＞あるいは＜ｐ２＞のようなｐｒｅｄｉｃａｔｅが付いた命令については、＜　＞内の変数がｐｓｅｑ命令より１にセットされた場合にのみ、その命令の結果がレジスタに反映される。一方、（５）のように、ｐｒｅｄｉｃａｔｅが付いていない命令については、常に、その命令の結果がレジスタに反映される。

【０００８】

したがって、図２８のプログラムコードにおいて、レジスタｒ３の値とレジスタｒ４の値とが等しいときは、ｐｓｅｑ命令によりｐ１に１、ｐ２に０がセットされ、このとき＜ｐ１＞が付いた命令“ｓｌｌ　ｒ６，ｒ１０，２”，“ｌｉ　ｒ５，１”と、ｐｒｅｄｉｃａｔｅが付いていない命令“ｍｏｖｅ　ｒ２，ｒ５”の結果だけがレジスタに反映されることになるため、図２７でＬ２に分岐した場合と同様の結果が得られる。同様に、レジスタｒ３の値とレジスタｒ４の値とが等しくないときは、ｐ１は０、ｐ２は１にセットされ、このとき＜ｐ２＞が付いた命令“ｌｉ　ｒ５，０”と、ｐｒｅｄｉｃａｔｅが付いていない命令“ｍｏｖｅ　ｒ２，ｒ５”の結果だけがレジスタに反映されることになるため、図２７

でＬ２に分岐しなかった場合と同様の結果となる。

【０００９】

ｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎの実施形態としては、ｐｒｅｄｉｃａｔｅの真偽によらず命令を実行し、真偽が判明した時点で真の命令の実行結果のみを状態に反映させる方法と、ｐｒｅｄｉｃａｔｅの真偽が判明するまで命令を実行せず、真と判明した命令のみを実行する方法があるが、制御依存する命令を並列に実行できるのは前者の方法だけである。

【００１０】

分岐予測方式は、条件分岐命令の分岐先を条件が決まる前に予測する方式である。予測方法としては、分岐予測先を静的に指定する方法（例えば、ループの繰り返しを判定するための条件分岐命令は、常に分岐すると予測する方法）や、専用のハードウェアを用いて動的に予測する方法（例えば、分岐命令ごとに前回の分岐方向を記録しておき、次回の分岐時には前回と同じ方向に分岐すると予測する方法）、これらの２つを組み合わせる方法があり、それぞれの方法に対して様々な実現手段が提案されている。

【００１１】

演算装置は予測先の命令を投機的に実行しておき、予測した分岐命令の条件が確定した時点で、予測が正しかったかどうかを判定する。予測が正しい場合には投機的に実行した命令の結果を演算装置の状態に反映させる。予測が誤っていた場合には投機的に実行した命令の結果は捨てられ、正しい分岐先に戻ってプログラムコードの実行を再開する。

【００１２】

ｄｅｐｅｎｄｅｎｃｅ　ｃｏｌｌａｐｓｉｎｇでは、データ依存する命令列を特殊な演算器で実行できる１命令に変換して実行する方式である。主に浮動小数点演算やマルチメディア演算に適用されている。

【００１３】

ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎは命令の入力が確定する前にその出力結果を予測する方式である。出力が予測された命令にデータ依存する命令列を予測された値を元に実行することで、データ依存する２つの命令列を並列に実行するこ

とができるようになる。

【００１４】

出力結果を予測する方法には、命令が前に出力した結果を記録しておき、その値を次回の予測値とする方法や、出力が特定の法則（例えば、出力値が一定の割合で増減しているという法則や、何種類かの出力値が一定の順序で繰り返し出力されているという法則など）によって変化しているような命令を発見し、その法則に従って出力を予測する方法等が提案されている。

【００１５】

ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎは現在の所はまだ研究段階であり、予測の適用率（予測を必要とする動的な命令数に対する、実際に予測が適用できた命令数の割合）、予測的中率（予測を適用した命令数に対する、予測が正しかった命令数の割合）ともにそれほど高くはなく、この手法を採用している一般的な演算装置は存在していない。

【００１６】

【発明が解決しようとする課題】

上述した依存性のある命令の並列実行方式のうち、まずｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎについては、分岐先の１方向に存在する命令は実行する必要がないにもかかわらず、両方向に存在する命令を実行しなければならないという問題がある。実行する必要のない命令により演算器が占有されると有効な命令の実行率が低下するため、全体の命令実行効率は低下してしまう。

【００１７】

また、ｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎにおいては、命令間の制御依存関係はｐｒｅｄｉｃａｔｅを介したデータ依存関係に置換されるため、ｐｒｅｄｉｃａｔｅをセットする命令列（変換前は分岐命令の条件をセットする命令列）とｐｒｅｄｉｃａｔｅを入力とする命令列（変換前は分岐先の命令列）は依然依存関係にあり、これらの命令列はプログラムコード上に依存順に配置される必要がある。よって、複数の分岐命令を越えた先の命令をｏｕｔ－ｏｆ－ｏｒｄｅｒ実行するためには、命令がｏｕｔ－ｏｆ－ｏｒｄｅｒ実行できるかどうかを判定する装置（例えば、スーパースカラでは命令ウィンドウ）に多くのエントリ

を持たせる必要が生じる。

【００１８】

分岐予測においては、ある分岐命令で分岐先の予測を誤ると、最悪の場合それ以後に実行される命令の全てが必要のない命令になってしまうという問題がある。個々の分岐命令の予測的中率は高くても、分岐予測が複数回連続して的中する確率は低くなるため、複数の分岐命令を越えた先にある命令の実行は、多くの場合無駄な命令実行になる。

【００１９】

また、制御依存する命令をプログラムコード上に並列に配置することはできないため、複数の分岐命令を越えた先の命令をｏｕｔ－ｏｆ－ｏｒｄｅｒ実行する場合には、ｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎと同様に規模の大きなハードウェアが必要となる。

【００２０】

ｄｅｐｅｎｄｅｎｃｅ　ｃｏｌｌａｐｓｉｎｇは、浮動少数点演算のような複雑な命令の演算時間を短縮する効果があるが、単純な命令ではさほど効果はない。また、変換後の命令を実行するための、専用の演算器が必要となる。

【００２１】

ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎでは、命令の出力結果を予測することにより、その命令以後の命令列と、その命令がデータ依存する命令列とを並列に実行できるようになる。しかしながら、これらの命令列は予測が正しかったかどうかを確認するために、プログラムコード上に元のデータ依存順に配置されなければならない。よって、ｐｒｅｄｉｃａｔｅｄ　ｅｘｅｃｕｔｉｏｎや分岐予測と同様、十分なｏｕｔ－ｏｆ－ｏｒｄｅｒ実行を行うには多くのエントリを持ったｏｕｔ－ｏｆ－ｏｒｄｅｒ実行判定装置が必要となる。

【００２２】

本発明は、上記事情を考慮してなされたもので、より小さなｏｕｔ－ｏｆ－ｏｒｄｅｒ実行判定装置を用いて、従来のハードウェアで実現されていたｏｕｔ－ｏｆ－ｏｒｄｅｒ実行より広い範囲でのｏｕｔ－ｏｆ－ｏｒｄｅｒ実行を可能とし、かつ不必要な命令実行が生じた場合においても、性能低下の度合を従来の手

法より小さくすることを可能とする中央演算装置及びコンパイル方法を提供することを目的とする。

【００２３】

【課題を解決するための手段】

本発明は、複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置（例えば、スーパースカラプロセッサ）において、前記プログラムが予め定められたデータ依存性を有する命令列（タスク）の単位で区切られ、各命令列間の制御依存性は所定の条件により該命令列を承認もしくは否認するコミット命令により表現され、該命令列間のデータ依存性は該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与することにより表現され、前記データ依存性を有する命令列が該命令列の単位で区切られたプログラム区間のいずれに属する命令列であるかを識別するための識別番号を該命令列に含まれる命令毎に付与する識別番号付与手段と、前記データ依存性を有する命令列の単位で命令を前記複数のバッファに割り当てる演算器割当手段と、特定の命令列が前記コミット命令の実行により承認された場合にのみレジスタを更新させるレジスタ更新手段（例えばグローバルレジスタ更新部）と、特定の命令列が前記コミット命令の実行により承認された場合にのみメモリを更新させるメモリ更新手段（例えばストアバッファ）とを備えたことを特徴とする中央演算装置を提供する。

【００２４】

ここで、前記識別番号付与手段は、直前の命令に付与した識別番号を保持する識別番号カウンタ手段と、識別番号を付与すべき命令が前記コミット命令以外の命令である場合には、前記カウンタ手段に保持されている識別番号と同じ識別番号を付与し、識別番号を付与すべき命令が前記コミット命令である場合には、前記識別番号カウンタ手段に保持されている識別番号を１だけ増加させた識別番号を付与する手段とを含むようにしてもよい。

【００２５】

また、前記演算器割当手段は、前記データ依存性を有する命令列が、前記複数

のバッファのいずれに割り当てられているかを示す情報を保持し、ある命令が属する前記データ依存性を有する命令列に対するバッファの割り当てが存在する場合には、前記情報を参照して該命令を該命令列に対応するバッファに割り当て、該命令列に対するバッファの割り当てが存在しない場合には、該命令が割り当てられた前記複数のバッファの中から、処理が行なわれていないバッファを選んで該命令を割り当てるようにしてもよい。

【００２６】

一方、前記演算器割当手段は、予め定められた規則により各データ依存性を有する命令列を前記複数のバッファに割り当てるようにしてもよい。

【００２７】

また、前記レジスタ更新手段は、前記複数のバッファから特定の命令が消去される際に、前記特定の命令を含む前記データ依存性を有する命令列が前記コミット命令の実行により承認された場合には、該特定の命令が特定のレジスタに退避した結果を用いてレジスタを更新し、該命令列が前記コミット命令の実行により否認された場合にはレジスタの更新を行なわないようにしてもよい。

【００２８】

また、前記メモリ更新手段は、前記演算実行ユニットで実行されたストア命令に関する情報を保持し、前記複数のバッファから前記ストア命令が消去される際に、前記ストア命令が含まれる前記データ依存性を有する命令列が前記コミット命令の実行により承認された場合には該ストア命令が登録した情報を用いてメモリを更新し、該命令列が前記コミット命令の実行により否認された場合にはメモリの更新を行なわないようにしてもよい。

【００２９】

また、前記フラグが付与されたレジスタを使用する命令をデコードする際に、該レジスタを使用不可とし、該レジスタが命令により更新された際に該レジスタを使用可とするようにしてもよい。

【００３０】

また、前記演算実行ユニットは、特定の命令の実行結果を命令の出力オペランドとは異なるレジスタに一旦退避し、該特定の命令が含まれる前記データ依存性

を有する命令列が前記コミット命令の実行により承認されたのちに、該レジスタの情報をレジスタ更新手段に送るようにしてもよい。

【００３１】

また、前記複数のバッファから該演算実行ユニットに送ることが可能な命令が複数存在する場合に、投機的でない命令を投機的な命令より優先させて該演算実行ユニットに送るようにしてもよい。

【００３２】

なお、上記した本発明にかかる中央演算装置は、中央演算装置における命令処理方法として実施することもできる。

【００３３】

さらに、本発明では、複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置で実行されるプログラムを生成するコンパイル方法であって、前記プログラムを予め定められたデータ依存性を有する命令列の単位で区切ったときに、該命令列のいずれに属する命令であるかを識別するための識別番号を命令毎に付与するとともに、該命令列間の制御依存性を表現するために、該命令列の承認もしくは否認を行なうコミット命令を発行し、該命令列間のデータ依存性を表現するために、該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与することを特徴とするコンパイル方法を提供する。

【００３４】

なお、上記した本発明にかかるコンパイル方法は、コンパイル装置として実施することもできる。

【００３５】

さらに、本発明では、複数の演算実行ユニットと該演算実行ユニットに属する複数のバッファを備え、プログラムの命令列を前記バッファに割り当てる中央演算装置で実行されるプログラムを生成するコンパイル手順であって、前記プログラムを予め定められたデータ依存性を有する命令列の単位で区切ったときに、該命令列のいずれに属する命令であるかを識別するための識別番号を命令毎に付与

するとともに、該命令列間の制御依存性を表現するために、該命令列の承認もしくは否認を行なうコミット命令を発行し、該命令列間のデータ依存性を表現するために、該命令列に含まれる命令がデータの生成側か消費側かを示すフラグを該命令が使用するレジスタに付与する手順を含むコンパイル手順をコンピュータで実行するためのコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体を提供する。

【００３６】

本発明では、ある命令の実行前に、その命令と制御依存関係、またはデータ依存関係にある命令を投機的に実行する命令処理方式において、データ依存する命令列（の各命令）に対して、データ依存列ごとに固有の識別番号を付加することにより、データ依存する命令列に属する命令をグループ（タスク）として管理することで、命令と命令の制御依存関係をプログラム上の配置位置に依らずに表現可能とし（該識別番号を持つ命令を条件により承認もしくは否認するコミット命令を発行することで制御依存関係の表現を可能とし）、さらに制御投機的に実行可能な命令を選択し実行するために必要なハードウェアを、従来の制御投機的命令実行方式に比較してより少なくすることができる（例えば、従来の命令ウィンドウよりも記憶容量の小さいハードウェアで、且つ、従来の投機的実行方式よりも小さな命令集合から実行可能な命令を選び出すことにより、効果的な制御投機的命令実行を可能とする）。

【００３７】

また、命令と命令のデータ依存関係を、データを生成する命令の出力レジスタ、またはデータを消費する命令の入力レジスタに、それぞれ生成フラグ、消費フラグを付加することにより、命令と命令のデータ依存関係をプログラム上の配置位置に依らずに表現可能とし（生成フラグ、消費フラグの付加された命令のフェッチ時に該レジスタを使用不可とし、レジスタの値が更新されたときに使用可とすることでデータ依存する命令間の実行順序を保証し）、さらにデータ投機的に実行可能な命令を選択し実行するために必要なハードウェアを、制御投機的実行と同様の理由により、従来のデータ投機的命令実行方式に比較してより少なくすることができる。

【００３８】

このように、本発明によれば、単純なハードウェアにより効果的な命令の投機的実行が可能となる。

【００３９】

また、これにより、単純なハードウェアが要求されるプロセッサにおいても投機的実行が可能になるという効果がある。また、投機的に実行可能な命令を選び出すためのハードウェアが小さいために、より多くの分岐命令を越える制御投機的実行や、より多くのデータ依存列を超えるデータ投機的実行が可能となり、一般的な方式による投機的実行より性能が向上するという効果がある。さらに、本発明は命令が投機的か否かをプログラム上に明示し、演算ユニットの使用に際しては投機的でない命令を優先させることにより、実際には実行する必要のない命令が投機的に実行されることによる性能低下を防ぐことが可能となるという効果がある。

【００４０】

【発明の実施の形態】

以下、図面を参照しながら発明の実施形態を説明する。

【００４１】

図１は、本実施形態に係るプロセッサ（中央演算装置）の構成例を示すブロック図である。

【００４２】

図１に示されるように、本実施形態に係るプロセッサは、メモリ１、アドレス生成部２、命令フェッチ部３、命令キュー４、命令デコード部５、タスクウィンドウ識別子生成部（識別番号付与手段）６、演算器割当部（演算器割当手段）７、実行キュー（バッファ）８～１０、オペランド状態判定部１１、ローカルレジスタ１５～１７、命令実行ユニット（演算実行ユニット）１２～１４、グローバルレジスタ１８、グローバルレジスタ更新部（レジスタ更新手段）１９、ロードバッファ２０、ストアバッファ（メモリ更新手段）２１を持つ。

【００４３】

なお、本プロセッサの持つ命令実行ユニット（１２～１４）の個数は一例であ

り、これに限定されるものではない。また、図１の例では、レジスタは全ての命令により使用できるグローバルレジスタ（１９）と、特定の演算ユニットに属する命令キューにある命令のみ利用できるローカルレジスタ（１５～１７）に分割されているが、グローバルレジスタのみを用いる実施形態も考えられる。

【００４４】

本実施形態では、コンパイラによりデータ依存すると見なされた命令列を「タスク」と呼ぶものとする。つまり、タスクに属する命令は、該命令よりプログラム上で前に配置された同タスクに属する命令に対してデータ依存関係にある。また、コミット命令の直後から始まり、コミット命令で終わる部分を「タスクウィンドウ」と呼ぶものとする。つまり、タスクウィンドウの終端の命令はコミット命令であり、かつ、そのタスクウィンドウにおいて唯一のコミット命令である。

【００４５】

また、本実施形態では、プロセッサ内において、各命令がどのタスクに属するかを識別するために、各命令には、それが属するタスクに固有の「タスク番号」がコンパイラにより付与される。また、各命令がどのタスクウィンドウに属するかを識別するために、各命令には、それが属するタスクウィンドウに固有の「タスクウィンドウ番号」がタスクウィンドウ識別子生成部により付与される。

【００４６】

図１のプロセッサの主なユニットの概要は次の通りである。

【００４７】

タスクウィンドウ識別子生成部６は、デコードされた命令に対して付与すべきタスクウィンドウ番号を生成するためのユニットで、少なくとも連続する２つのタスクウィンドウに対して異なる識別子を付与する。タスクウィンドウ識別子生成部６は例えば最大値が２以上のループカウンタで実現できる。

【００４８】

演算器割当部７は、デコードされた命令をどの実行キューに挿入するかを決定して、命令を挿入するユニットである。実行キューからの命令実行は、実行キューに命令が挿入された順序で行われる。

【００４９】

オペランド状態判定部１１は、デコードされた命令が使用するオペランドが使用可能かどうかを判定するためのユニットである。

【００５０】

グローバルレジスタ更新部１９は、コミット命令により承認された命令によるグローバルレジスタの更新が、１命令ずつ行われるよう制御するためのユニットである。

【００５１】

ロードバッファ２０は、ロード命令が依存するストア命令がストアバッファに登録されているかどうかを求め、該ストア命令がストアバッファに登録されている場合にはストアバッファの値をロードし、登録されていない場合にはデータキャッシュおよびメモリから値をロードするユニットである。

【００５２】

ストアバッファ２１は、実行されるストア命令に関する情報（ストアする値やアドレスなど）を保存しておき、該ストア命令がコミット命令により承認された後に、保存された情報に基づいて、ストアバッファの登録順にメモリを更新するユニットである。

【００５３】

本実施形態のプロセッサは、概略的には、命令をタスク単位で実行キューに登録し、実行キューからの実行は登録順に行ない、グローバルレジスタやメモリを更新する命令の結果はローカルレジスタやストアバッファに一時保存して、コミット命令により承認された命令の演算結果のみをプロセッサの状態に反映させるものであり、データ依存・制御依存にともなう命令の配置位置の制約をなくすことで並列度を抽出するために必要なバッファのサイズを小さくすることを可能とし、且つ実行可能な命令を命令キューの先頭の命令のみから選び出すことで命令の選択処理時間を短縮してより多くの分岐命令やデータ依存列を越える命令を投機的に実行することができる。

【００５４】

次に、図２に、本実施形態のプロセッサにおけるパイプライン処理の流れの一例を示す。

【００５５】

命令フェッチ部３は、アドレス生成部２により示される命令アドレスをアドレスバス２２で指定し、その命令から始まる複数の命令を命令キャッシュまたはメモリから命令フェッチバス２３を介してフェッチする（ステップＳ１１）。フェッチされた命令は、命令キューバス２４を介して命令キュー４にフェッチ順に挿入される。

【００５６】

命令フェッチ部３は、命令キューが一杯になった場合には命令キューに空きができるまで次のフェッチを停止する。

【００５７】

アドレス生成部２は、通常は最後にフェッチした命令の次を指すように更新される。ただし、１．ジャンプ命令が承認されたとき、２．１ｃｍｔ命令をフェッチしたときには、アドレス生成部２は各命令が指定するアドレスを指すように更新される。

【００５８】

命令デコード部５は、命令キュー４の先頭から、複数の命令をデコードバス２５を介してデコードし、各命令についてその種類と使用するオペランド、消費・生成フラグおよび命令のタスク番号を得る（ステップＳ１２）。さらに、タスクウィンドウ識別子生成部６から各命令のタスクウィンドウ番号を得て（ステップＳ１３）、これらの情報を演算器割当部７に送る。

【００５９】

演算器割当部７は、命令デコード部から送られた情報を用いて命令を挿入する実行キューを選択し、実行キューバス２６～２８を介して挿入する（ステップＳ１４）。挿入すべき実行キューが一杯の場合には、実行キューに空きができるまで待ち、その間は命令デコード部による命令のデコードを停止させる。また、実行キューにまだ実行されていないコミット命令が存在している状態で、新たなコミット命令をデコードした場合にも同様の処理を行なう。

【００６０】

演算器割当部７は、命令が使用する命令実行ユニットを命令アドレスから得る

。その命令実行ユニットに属する実行キューの中から、命令を挿入する実行キューを選択する方式については、いくつかの実現例が考えられる。

【００６１】

例えば、演算器割当部７における実行キューの選択方式としては、図３のように、タスク番号と実行キューのマップを保持するメモリを持たせて、命令のタスク番号に対するマップが存在する場合には、該マップで示される実行キューに命令を挿入し、マップが存在しない場合には命令が使用する命令実行ユニットに属する実行キューの中から空の実行キューを選択して挿入し、タスク番号と実行キューとのマップを生成するものが考えられる。コミット命令によりタスクが承認・否認されると、そのタスクに対するマップは破棄される。

【００６２】

命令の実行キューへの割当をコンパイラで行なう場合には、命令デコード部５はデコードの際に命令を挿入すべき実行キューに関する情報を得る。この場合には、演算器割当部７はこの情報に従って実行キューを選択して命令を挿入する。

【００６３】

図４に実行キューの構成例を示す。この例では、実行キューはリングバッファとして構成されている。実行キューには、１．ｈｅａｄ、２．ｔａｉｌ、３．ｅｘｅｃｕｔｅ、４．ｃｏｍｍｉｔの４つのポインタがある。ｈｅａｄポインタ、ｔａｉｌポインタはリングバッファに登録されているエントリの先頭と終端を示すポインタである。ｅｘｅｃｕｔｅポインタは次に実行すべきエントリを示すポインタで、命令が実行されると次のエントリを指すように移動される。ｃｏｍｍｉｔポインタは次にグローバルレジスタ・メモリを更新する命令を指すポインタで、更新が完了すると次のグローバルレジスタ・メモリを更新する命令を指すように移動される。

【００６４】

実行キューの各エントリは、命令の種類、使用するオペランド、例外情報を記録するフィールドを持つ。承認・否認されていない命令が例外を起こした場合には、その例外に関する情報が対応するフィールドに記録され、例外処理は承認後まで遅らされる。

【００６５】

オペランド状態判定部１１は実行キュー８～１０の先頭の命令が使用するオペランドが使用可能かどうかを判定した（ステップＳ１５）後、オペランドが使用可能で且つその命令を実行する命令実行ユニット（キューの属する命令実行ユニット）に空きがあれば命令を対応する命令バスを介して転送し、オペランドをグローバルレジスタ１８およびローカルレジスタ１５～１７からフェッチした後、命令が実行される（ステップＳ１６）。

【００６６】

オペランド状態判定部１１は、図５に示すようなレジスタスコアボードによりオペランドが使用可能か否かを判定する。レジスタに生成フラグまたは消費フラグのついた命令がデコードされると、レジスタスコアボードの該レジスタに対するｖａｌｉｄビットが０になり、該レジスタの値が更新されるとｖａｌｉｄビットが１になる。オペランド状態判定部は、命令が使用する全てのオペランドに関して、レジスタスコアボードにおけるｖａｌｉｄビットが１であれば命令は実行可能と判定される。

【００６７】

実行可能な命令の一部は、１．命令実行ユニットの数、２．グローバルレジスタのリードポート数、３．データキャッシュのリードポート数、４．ストアバッファのライトポート数、５．ストアバッファのエントリ数、６．ロードバッファのエントリ数といったリソースの制約により実行できない場合がある。オペランド状態判定部１１は、実行可能な命令の中から予め定められた規則に従って実行すべき命令を選び、命令実行ユニットに送る。

【００６８】

命令選択規則の一例としては、同命令実行ユニットに属する命令の間では最も古く実行キューに入った命令を優先し、他の命令実行ユニットに属する命令との間では最も優先度の高い命令実行ユニットに属する命令を優先するという規則が考えられる。

【００６９】

また、命令を挿入する実行キューをコンパイラで指定している場合には、リソ

ースの制約を満たすように命令を実行キューに割り当てることで、所属する命令実行ユニットの異なる命令間の優先順位を考慮する必要がなくなる。例えば、ストアバッファのライトポートが１つの場合には、ストア命令を特定の命令実行ユニットに属する実行キューにのみ割り当てれば、ストアバッファのライトポートに関するリソース制約を調べる必要はない。

【００７０】

さらに、命令の実行キューへの割当をコンパイラで行なうか否かにかかわらず、投機的なタスクには特定のタスク番号を割り当て、同命令実行ユニットに属する命令の間では、投機的でない命令を投機的な命令に優先して実行するように制御するという命令選択規則も考えられる。この規則を用いると、実行する必要のない命令が投機的に実行されることによる性能低下を防ぐことができる。

【００７１】

命令実行の際に例外が発生した場合には、例外の種類をその命令が存在する実行キューのエントリに記録し、その時点では例外処理は行なわない。

【００７２】

命令の実行後、その結果はレジスタを出力とする命令の場合にはローカルレジスタに出力され（ステップＳ２０）、メモリを出力とする命令（ストア命令）の場合にはその情報がストアバッファ２１に登録される（ステップＳ２１）。

【００７３】

そして、コミット命令が実行されると、コミット命令は指定された条件に従って一部のタスクを承認し、また一部のタスクを否認する（ステップＳ１９）。

【００７４】

グローバルレジスタを出力とする命令が承認されると、その命令の結果を保持するローカルレジスタの値と、グローバルレジスタ番号がグローバルレジスタ更新部１９に送られる。レジスタ更新部のライトポート数を超える命令が承認された場合には、予め定めた優先順位に従って登録する。この優先順位としては、例えば前記のオペランド状態判定部における優先順位がある。レジスタ更新部は登録された情報を用いて、パイプラインとは独立に１サイクルに一定数（グローバルレジスタのライトポート数）のグローバルレジスタの値を登録された順序で更

新する。

【００７５】

各命令実行ユニットは２つのローカルレジスタセットを持っており、タスクウィンドウ番号が偶数のタスクウィンドウに属する命令と、奇数のタスクウィンドウに属する命令は異なるローカルレジスタセットを使用する。

【００７６】

一方、メモリを出力とする命令が承認されると、ストアバッファは登録された情報を用いて、パイプラインとは独立に一定数（データキャッシュのライトポート数）のメモリの値を登録された順序で更新する。

【００７７】

例外の種類が記録されたエントリに存在する命令が承認されると、例外の種類に従った例外処理が行なわれる。

【００７８】

ローカルレジスタを出力とする命令や出力を持たない命令は、承認・否認後に何も処理を行なわないまま実行キューから消去される。実行キューからの消去は、単に実行キューのｈｅａｄポインタを移動することにより実現される。

【００７９】

次に、図６に、本実施形態のプロセッサにおいてロード命令、ストア命令を実行した際の動作の一例を示す。

【００８０】

ストア命令は、１．ストアバッファが一杯の場合、２．タスクウィンドウ番号が等しいストア命令がストアバッファに存在し、かつ前記ストア命令とストアバッファのストア命令の属するタスクウィンドウが異なる場合には実行されない。

【００８１】

まず、ストア命令は命令実行ユニットに送られると、ストアするメモリアドレスが計算される（ステップＳ１１）。次に、命令の種類が判定されたのち（ステップＳ１２）、ストア命令に関する情報がストアバッファ２１に登録される（ステップＳ１３）。

【００８２】

図７にストアバッファの構成例を示す。

【００８３】

この構成例では、ストアバッファはリングバッファとして構成されている。ストアバッファの各エントリは、１．ストアするアドレス、２．ストアする値、３．ストア幅（ワードデータ、バイトデータなど）、４．ストア命令が属するタスクウィンドウ番号、５．ｃｏｍｍｉｔビット、６．ｃｌｅａｒビットを持つ。

【００８４】

ストア命令がストアバッファに登録されると、そのエントリに上記要素がセットされる。ｃｏｍｍｉｔビット、ｃｌｅａｒビットは０にセットされる。

【００８５】

コミット命令によりストア命令が承認されると（ステップＳ１４）、該ストア命令が登録されているストアバッファのエントリのｃｏｍｍｉｔビットが１にセットされる（ステップＳ１６）。ストア命令が否認されると（ステップＳ１４）、該ストア命令が登録されているストアバッファのエントリのｃｌｅａｒビットが１にセットされる（ステップＳ１５）。

【００８６】

ストアバッファの先頭エントリのｃｏｍｍｉｔビットが１になったら、そのエントリの情報を用いてメモリを更新し、ストアバッファの先頭を示すｈｅａｄポインタを次のエントリに移動する（ステップＳ１８）。一方、先頭エントリのｃｌｅａｒビットが１になった場合には、単にｈｅａｄポインタを次のエントリに移動するだけで、メモリの更新は行なわない（ステップＳ１７）。

【００８７】

次に、ロード命令の実行例について説明する。

【００８８】

オペランド状態判定部１１は、ロード命令に関しては前記命令が実行可能となる条件に加えて、「前のタスクウィンドウに存在する全てのストアは実行済である（ストアバッファへの登録が完了している）か、コミット命令により否認されている」という条件を満たすときに、ロード命令は実行可能であると判定する。

【００８９】

ロード命令が命令実行ユニットに送られると、ロードするメモリアドレスが計算される（ステップＳ１１）。次に、命令の種類が判定されたのち（ステップＳ１２）、ストアバッファが参照され、ロードがメモリ依存するストア命令がストアバッファに登録されているかが調べられる（ステップＳ１９）。

【００９０】

ロード命令がストアバッファに登録されているストア命令と依存関係を持つかどうかは、予め定められた基準により判断する。コンパイラはロード命令とストア命令の依存関係がこの基準に従って表現されるように命令スケジュールを行なう。

【００９１】

ロードとストア命令の依存関係を示す基準の一例としては、ロード命令は、ロードする領域を更新するストア命令のうち、１．該ロード命令が属するタスクウィンドウより前のタスクウィンドウに属するストア命令、または、２．該ロード命令と同じタスクに属し、該ロード命令より前に配置されているストア命令のいずれかの条件を満たすストア命令と依存するという基準が考えられる。

【００９２】

ロード命令と依存するストア命令がストアバッファに存在した場合には、最も新しくストアバッファに登録されたストア命令がストアする値（ストア幅がロード幅より大きい場合には、ストアする値の一部）をロード命令の出力として用いる（ステップＳ２１）。一方、ロード命令と依存するストア命令がストアバッファに存在しない場合には、データキャッシュを参照する（ステップＳ２２）。

【００９３】

ロードする領域がデータキャッシュにヒットした場合には（ステップＳ２３）、キャッシュの値をロードしてロード命令は完了する（ステップＳ２４）。一方、データキャッシュミスやページフォルト例外が生じたロード命令は、ロードバッファ２０に登録される（ステップＳ２５）。

【００９４】

ロードバッファの構成例を図８に示す。

【００９５】

この構成例では、ロードバッファはリングバッファとして構成され、ロードバッファの各エントリは、１．ロードするアドレスの下位数ビットを０にマスクしたアドレス（キャッシュアドレスと呼ぶ）、２．複数のレジスタ番号からなるフィールド（レジスタフィールド）、３．ｖａｌｉｄフィールド、４．ｒｅｆｉｌ１ビットを持つ。登録するレジスタ番号の数はデータキャッシュの１ラインに含まれるワード数で、マスクのビット数は２のｎ乗がキャッシュラインのバイト数に等しくなるようなｎである。図８は、キャッシュラインが８ワードの場合を示している。

【００９６】

レジスタフィールドはキャッシュラインのワードがＭＳＢから順にレジスタフィールドに示されるレジスタにロードされることを示し、ｖａｌｉｄフィールドはＭＳＢから順に、レジスタフィールドに示されるレジスタ番号が有効であるか否かを示す（１なら有効）。無効なレジスタに対するロードは行われない。

【００９７】

ロードバッファへのロード命令の登録手順の例を図９に示す。

【００９８】

まず、ロード命令のキャッシュアドレスと等しいアドレスが登録されているエントリがロードバッファに存在するか否かが調べられ（ステップＳ１１，１２）、存在する場合には最も新しいエントリのｖａｌｉｄフィールドを用いて、ロード命令の出力レジスタを登録するレジスタフィールドが有効か否かを調べる（ステップＳ１６）。

【００９９】

ロード命令のキャッシュアドレスと等しいアドレスを持つエントリがない場合、または、エントリはあるが該ロード命令の出力レジスタをレジスタフィールドに登録できない場合（登録するレジスタフィールドが既に有効である場合、または該エントリのｒｅｆｉｌｌビットが１の場合）には、ロードバッファに新たなエントリを作ってロード命令を登録する（ステップＳ１５）。ロード命令の登録の際には、ｒｅｆｉｌｌビットは０に設定される。

【０１００】

一方、該ロード命令のキャッシュアドレスを持つロードバッファのエントリが存在し、且つ該ロード命令の出力レジスタを登録する該エントリのフィールドが有効でない場合には、該エントリの対応するレジスタフィールドに該ロード命令の出力レジスタ番号を登録し、該エントリの対応するｖａｌｉｄフィールドに１を立てる（ステップＳ１４）。

【０１０１】

ロードバッファの先頭エントリにあるｖａｌｉｄフィールドのいずれかのビットに１が立っていて、且つ該エントリのｒｅｆｉｌｌビットが０の場合には、該エントリのキャッシュアドレスの内容がメモリから読み出される。読み出しが完了したら、該エントリのｒｅｆｉｌｌビットを１とする。

【０１０２】

該エントリのｒｅｆｉｌｌビットが１になったら、レジスタフィールドに登録された有効なレジスタへのロードを、Ｒ１から順に行なう。有効な全てのレジスタへの値のロードが完了したら、ロードバッファのｈｅａｄポインタを次のエントリに移動する。

【０１０３】

本実施形態では、ロード命令の種類として、１．実行可能になり次第実行できるロードと、２．実行可能になってもコミット命令により承認されるまでは実行できないロード（ｄｅｌａｙｅｄ－ｌｏａｄ）の２種類を持たせる実施形態も考えられる。ｄｅｌａｙｅｄ－ｌｏａｄは該コミット命令の次のタスクウィンドウに属するとすることで、ロード命令とストア命令の依存関係（メモリ依存関係）の表現に伴う命令配置の制約を軽減することができる。

【０１０４】

つまり、メモリ依存関係により特定のタスクウィンドウに配置せざるを得ないロード命令は、ｄｅｌａｙｅｄ－ｌｏａｄに変更することで該タスクウィンドウより前のタスクウィンドウに配置することが可能となる。

【０１０５】

ｄｅｌａｙｅｄ－ｌｏａｄには、コード密度の向上や命令のプリフェッチによる命令キャッシュミスペナルティの緩和といった効果がある。

【０１０６】

また、本実施形態では、ロード命令の種類として、依存性が曖昧なストア命令とは依存関係を持たないとするロード（ＭＤ－ｌｏａｄ）を持たせる実施形態も考えられる。ＭＤ－ｌｏａｄを実行する際にはストアバッファの参照は行なわれず、直接データキャッシュやメモリの内容がロードされる。また、ＭＤ－ｌｏａｄに関する情報が、メモリコンフリクトバッファに登録される。

【０１０７】

図１０にメモリコンフリクトバッファの構成例を示す。

【０１０８】

この構成例では、メモリコンフリクトバッファのエントリは１．ＭＤ－ｌｏａｄのタスク番号、２．ＭＤ－ｌｏａｄがロードするアドレス、３．ロード幅（ワード、バイトなど）、４．ｃｏｎｆｌｉｃｔビット、５．ｖａｌｉｄビットを持つ。ＭＤ－ｌｏａｄの登録の際には、ｃｏｎｆｌｉｃｔビットは０、ｖａｌｉｄビットは１にセットされる。

【０１０９】

オペランド状態判定部１１は、ストア命令が実行可能か否かを判定する際に、該ストア命令が属するタスクウィンドウより前のタスクウィンドウに属するＭＤ－ｌｏａｄ命令が、実行されていない、且つコミット命令により承認・否認が行なわれていない場合には、該ストア命令を実行可能としない。

【０１１０】

ストア命令が命令実行ユニットに転送され、ストアするアドレスが計算されると、コンフリクトバッファが探索されて、ストアアドレスと重なりのある領域をロードするＭＤ－ｌｏａｄ命令が求められ、このようなＭＤ－ｌｏａｄ命令が存在する場合には、該ＭＤ－ｌｏａｄ命令が登録されたエントリのｃｏｎｆｌｉｃｔビットが１にセットされる。

【０１１１】

ＭＤ－ｌｏａｄ命令は、コミット命令により承認・否認される前に、ｃｃｍｔ命令により、依存関係がないと見なしたストア命令との間に実際には依存関係があったかどうかを確認する必要がある。コンパイラは、ＭＤ－ｌｏａｄ命令を含

むタスクが承認される前に、必ずｃｃｍｔ命令が実行されるように命令スケジュールを行なう必要がある。

【０１１２】

ｃｃｍｔ命令はコミット命令の一種であり、ソースオペランドとしてタスク番号を指定する。ｃｃｍｔ命令は、コンフリクトバッファから指定されたタスク番号を持つ有効な（つまり、ｖａｌｉｄビットが１である）エントリを求めてｖａｌｉｄビットを０とする。また、該エントリのｃｏｎｆｌｉｃｔビットを調べて、ｃｏｎｆｌｉｃｔビットが１の場合には、指定されたタスクを先頭から再実行する。

【０１１３】

タスクの再実行によりプロセッサの正しい状態が得られることを保証するために、コンパイラは、ＭＤ－ｌｏａｄ命令を含むタスクが使用する入力オペランドが、ＭＤ－ｌｏａｄ命令の属するタスクウィンドウ内で使い回されないようにレジスタ割当をする必要がある。また、一つのタスクには高々一つのＭＤ－ｌｏａｄ命令が含まれるようにタスク割当をしなければならない。

【０１１４】

以上のような機構により、依存性が曖昧なストア命令を越えてロード命令を前に移動することが可能となる。

【０１１５】

次に、本実施形態における、命令間の依存関係を表現する方法の一例を示す。

【０１１６】

まず、この表現方法の例では、命令間の制御依存関係は、１．ｃｍｔ命令、２．ｂｃｍｔ命令、３．ｌｃｍｔ命令の３種類のコミット命令で表現される。

【０１１７】

ｃｍｔ命令は、条件式、承認するタスクの集合、条件式が満たされる際に否認するタスクの集合を入力とする。

【０１１８】

図１１に条件分岐命令を用いた従来のプログラムコードの例を示す。ｂｎｅ命令は、２つの入力オペランドの値が異なるときに、指定のアドレスに分岐する命

令である。

【０１１９】

図１１のプログラムコードを、ｃｍｔ命令を用いて表現した例が図１２である。各命令が属するタスク番号は命令の前に付加されている。ｃｍｔ命令が実行されると、指定された条件式（図１２の例では＄３の値と＄６の値は等しくない）が調べられる。条件式が満たされない場合には、承認するタスクの集合（図１２の例ではタスク１，２，３）に属するタスクが承認される。条件式が満たされない場合には、承認するタスクの集合に属するタスクのうち、否認するタスクの集合（図１２の例ではタスク２，３）に属さないタスクが承認され、否認するタスクの集合に属するタスクは否認される。

【０１２０】

否認されたタスクに属する命令の実行結果はプロセッサの状態に反映されないため、＄３と＄６が等しくない場合には、図１１におけるｂｎｅ命令の次の命令から、ラベル＄Ｌ４８の直前までの命令は実行されない。このようにして、ｃｍｔ命令により条件分岐が実現される。

【０１２１】

ｂｃｍｔ命令は、条件式、承認するタスクの集合、条件式が満たされる際に否認するタスクの集合、条件式が満たされる際に分岐する分岐アドレスを入力とする。

【０１２２】

図１３に条件分岐命令を用いた従来のプログラムコードを示し、該プログラムコードを、ｂｃｍｔ命令を用いて表現した本実施形態のプログラムコード例を図１４に示す。

【０１２３】

本実施形態においては、従来のプログラムコードにおいて、条件分岐命令で分岐が生じない場合に実行される全ての命令が，該条件分岐命令に対応するコミット命令より前に配置される場合には、条件分岐はｃｍｔ命令により表現する。一方、前記全ての命令の一部を該コミット命令より前に配置しない場合にはｂｃｍｔ命令を用いて、条件が満たされた場合には分岐をすることで該コミット命令よ

り前に配置しなかった命令を実行しないようにする。

【０１２４】

ｂｃｍｔ命令が用いられる例としては、１．依存性の表現による制約によりコミット命令を越えて命令を前に移動できない場合や、２．コミット命令を越える命令移動を行わないほうが効率よく実行できる場合などが挙げられる。

【０１２５】

図１４の例は、ｊ命令（直接ジャンプ命令）がコミット命令を越えて移動できない実施形態の例である。ｊ命令によるプログラムカウンタの変更は承認後に行ない、承認されるまではプログラムカウンタを１ずつ増大させるような実施形態では、ｊ命令をコミット命令より前に移動することが可能である。

【０１２６】

ｌｃｍｔ命令は条件式、ループを構成するタスク番号の集合、１回目のループでのみ承認するタスク番号の集合、および条件式が満たされる際に分岐する分岐アドレスを入力とする。ｌｃｍｔ命令はループ構造で現れる制御依存性を表現するのに用いられる。

【０１２７】

図１５に条件分岐命令を用いた従来のプログラムコードを示し、該プログラムコードを、ｌｃｍｔ命令を用いて表現した本実施形態のプログラムコード例を図１６に示す。

【０１２８】

ｌｃｍｔ命令が最初に実行されると、まずループを構成するタスク（図１６では３と４）、および１回目のループでのみ承認するタスク（図１６では１と２）が承認される。

【０１２９】

ｌｃｍｔ命令の条件式が成立する場合には、該ｌｃｍｔ命令で指定された分岐アドレス（ループの先頭）へ分岐する。このとき、ループを構成するタスクのタスク番号は可能な限り未使用のタスク番号にリネーミングされる。タスク番号が変更されると、コンパイラがタスクを実行キューに割り当てる実施形態（タスク番号と実行キューが静的に対応している実施形態）においても、タスクが挿入さ

れる実行キューは１回前のループでタスクが挿入された実行キューとは異なる。

【０１３０】

２回目以降にｌｃｍｔ命令が実行される場合には、ループを構成するタスクはタスク番号のリネーミング情報を用いて承認され（例えば、図１６で３と４がそれぞれ５と６にリネーミングされていたら、５と６が承認される）、１回目のループでのみ承認するタスクは否認される。

【０１３１】

タスク番号のリネーミング情報はループを１回実行するごとに更新され、ｌｃｍｔ命令の条件式が成立しなくなった時点で、該リネーミング情報は破棄される。

【０１３２】

一般的なスーパースカラプロセッサではループを繰り返し実行する際の命令並列度を向上させるために、命令が出力するレジスタをダイナミックに変更するレジスタリネーミング機構を持っている。本実施形態では、命令が使用するローカルレジスタのセットは、該命令が属するタスクウィンドウによって分けられているため、レジスタリネーミング機構なしにループ実行時の命令並列度を向上させることが可能である。

【０１３３】

次に、本実施形態においてデータ依存性がどのように表現されるかについて説明する。

【０１３４】

ある命令（命令Ｂとする）が別の命令（命令Ａとする）に対してデータ依存関係を持つ（命令Ａの出力結果を命令Ｂが使用する）場合、命令Ａを命令Ｂより前に配置したいときには、命令Ａの出力オペランドに生成フラグを付与する（図１６では命令の出力オペランドにＰを付与することで示している）。命令Ｂを命令Ａより前に配置したいときには、命令Ｂの入力オペランドのうち、命令Ａの結果に対応する方の入力オペランドに消費フラグを付与する。

【０１３５】

本発明の実施形態としては、前記命令Ｂが使用する前記命令Ａの結果が得られ

る前に、命令Ｂの演算結果を予測して実行する（ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎと呼ぶ）機構を備える形態も考えられる。前記命令Ａの結果を予測するための機構としては、例えばｌａｓｔ　ｏｕｔｃｏｍｅ－ｂａｓｅｄ　ｖａｌｕｅ　ｐｒｅｄｉｃｔｏｒなどがある（“Ｈｉｇｈｌｙ　Ａｃｃｕｒａｔｅ　Ｄａｔａ　Ｖａｌｕｅ　Ｐｒｅｄｉｃｔｉｏｎ　ｕｓｉｎｇ　Ｈｙｂｒｉｄ　Ｐｒｅｄｉｃｔｏｒｓ”，ＩＥＥＥ　Ｍｉｃｒｏ　’９７）。

【０１３６】

本実施形態は、ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なうためのコミット命令として、ｄｃｍｔ命令を備える。ｄｃｍｔ命令は、レジスタ番号と、タスク番号を入力とする。

【０１３７】

図１７にｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なうためにコンパイラが生成するプログラムコードの例を示す。

【０１３８】

コンパイラは、ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行ないたい命令がタスクの先頭になるようにタスク割当を行ない、該タスクがｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なうタスクであることをプログラムコード上に明示する（図１７では命令ニーモニックの前にｄ．をつけて表現している）。また、該タスクが承認される前に、該命令の出力オペランドとそのタスク番号を入力とするｄｃｍｔ命令が実行されるように命令スケジュールを行なう。

【０１３９】

予測した値が正しかったかどうかを判定するために、該命令と同じ命令を、ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なわないタスクにも割り当てる。

【０１４０】

オペランド状態判定部は、ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なうタスクの先頭命令に対しては入力オペランドが使用可能か否かは判定せずに、該命令を出力結果予測部に転送する。そして、出力結果予測部から得られた値を該命令の結果としローカルレジスタに出力する。

【０１４１】

ｄｃｍｔ命令を実行するときには、入力レジスタの値が予測された値と等しいかどうかが判定され、等しい場合には入力タスクが承認される。等しくない場合には、前記ｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なうタスクを、予測を行なわないタスクとして再実行する。このときには、ｄｃｍｔ命令は処理を行なわない。

【０１４２】

また、本実施形態においては、サブルーチン間の依存関係はｊｃｍｔ命令とｒｃｍｔ命令を用いて表現される。

【０１４３】

ｊｃｍｔ命令は呼び出すサブルーチンのアドレス、およびタスク番号の集合を入力とする。

【０１４４】

ｊｃｍｔ命令は、サブルーチンから復帰した後に実行を開始するアドレスを結果として出力する。また、入力として与えられたタスクを承認する。そして、指定されたサブルーチンに制御を移す。

【０１４５】

ｒｃｍｔ命令はレジスタ番号とタスク番号の集合を入力とする。ｒｃｍｔ命令が実行されると、指定されたタスクが承認される。そして、指定されたレジスタのアドレスに制御を移す。

【０１４６】

さて、以下では、本実施形態のプロセッサにおいて、命令が実行される際に各ユニットがどのように動作するかについて、およびいくつかのユニットの構成もしくは機能について、より詳しく説明する。

【０１４７】

以下の説明において、各パイプラインステージで行なわれる処理を次のように定義する。

・フェッチステージ（Ｆステージ）…メモリから命令をフェッチする

・デコードステージ（Ｄステージ）…フェッチした命令をデコードし、実行キューに挿入する

・実行ステージ（Ｅステージ）…命令実行ユニットを用いた演算を行なう
・メモリステージ（Ｍステージ）…キャッシュからのデータの読み出し、およびストアバッファへの登録を行なう
・ライトバックステージ（Ｗステージ）…演算結果をローカルレジスタに書く

また、以下では、「命令ｉ」は命令番号ｉの命令を表すこととする。

【０１４８】

図１８に、本プロセッサで実行するプログラムの一例を示す。

【０１４９】

以下、図１８のプログラムを実行対象とした場合を例にとってサイクルの流れに沿って説明する。

【０１５０】

なお、説明を簡明にするため、図１８のプログラムの開始時点において、パイプラインで処理中の命令は存在しないものとする。

【０１５１】

まず、サイクル１に、Ｆステージで命令フェッチ部３は命令１～３（命令番号は命令の末尾に示している）を順にフェッチして命令キュー４に挿入する。本発明は同時フェッチ数がいくつのプロセッサであっても適用可能であるが、本実施形態では同時フェッチ数を３としている。

【０１５２】

次に、サイクル２に、Ｆステージで命令フェッチ部３は命令４～６をフェッチして命令キュー４に挿入する。

【０１５３】

一方、サイクル２にＤステージでは、命令デコード部５は命令キュー４にある命令１～３をデコードし、命令の種類と使用するオペランド、タスク番号および生成・消費フラグを得る。また、各命令はタスクウィンドウ識別子生成部６からタスクウィンドウ番号を得る。

【０１５４】

演算器割当部７は命令デコード部５から送られたタスクウィンドウ番号から、命令１～３をそれぞれ実行キュー８、９、１０に挿入する。また、オペランドと

生成・消費フラグの情報から、グローバルレジスタ番号５および命令実行ユニット１２のローカルレジスタ番号５、命令実行ユニット１３のローカルレジスタ番号１を使用不可とする。

【０１５５】

次に、サイクル３に、オペランド状態判定部１１は各実行キューの先頭の命令が実行可能か否かを判定し、命令１～３は実行可能なので対応する命令実行ユニットに転送されて、命令が実行される。命令１、命令２の出力はローカルレジスタなので、これらのレジスタは命令実行の完了後、使用可になる。

【０１５６】

一方、サイクル３にＤステージでは、命令デコード部５は、命令４～６をデコードして、実行キューへの挿入を行なう。命令４～６が使用するオペランドには生成・消費フラグが付与されていないので、特定のレジスタを使用不可とする処理は行なわれない。Ｆステージでは、命令フェッチ部３は命令７～９をフェッチする。以後のサイクルにおいても、特に断りがない場合にはＦステージ、Ｄステージで同様の処理が行なわれているものとする。

【０１５７】

サイクル４にＥステージで命令４が実行キューに送られ、ロードアドレスの計算が行なわれる。命令４が使用するローカルレジスタ５番の値は、命令１の演算結果をバイパスすることで求められる。

【０１５８】

サイクル５に、Ｗステージで命令１、命令２の結果がローカルレジスタに出力される。命令３の結果も、グローバルレジスタに出力する結果を退避するためのローカルレジスタに出力される。

【０１５９】

一方、サイクル５にＭステージでは命令４のロードアドレスを用いてデータキャッシュが参照される。ここでキャッシュミスが生じたとすると、命令４はロードバッファ２０に登録される。

【０１６０】

命令７はローカルレジスタ１番が使用不可なためサイクル５では実行されない

【０１６１】

サイクル６にロードバッファでは命令４のロードアドレスに対するデータキャッシュのｒｅｆｉｌｌ処理が開始される。このとき、ｒｅｐｌａｃｅされるデータキャッシュのエントリはｉｎｖａｌｉｄａｔｅされる。

【０１６２】

一方、サイクル６にＥステージでは命令１１、命令１２が実行されロードアドレスの計算が行なわれる。命令１０はローカルレジスタ２番が使用できないため実行されない。

【０１６３】

サイクル６にＦステージで命令１６～１８がフェッチされる。命令１４のコミット命令がまだ実行されていない状態で新たなコミット命令（命令１８）をフェッチしたので、命令フェッチ部３は命令１４が実行キューからなくなるまでは命令フェッチを停止する。

【０１６４】

サイクル７にMステージでは命令１１と命令１２のロードアドレスを用いてデータキャッシュが参照される。命令１２はキャッシュヒットしたとすると、キャッシュから値が読み出される。一方、命令１１はキャッシュミスし、命令１１のロードアドレスと命令４のロードアドレスとが同キャッシュラインで異なるアドレスとすると、命令１１は命令４のロードバッファエントリにマージされる。

【０１６５】

一方、サイクル７にＥステージでは命令１５が実行され、命令１３、命令１４は実行されない。

【０１６６】

サイクル８にWステージで命令１２により命令実行ユニット１２のローカルレジスタ３番の値が更新される。

【０１６７】

一方、サイクル８にＤステージでは命令１９～２１がデコードされる。命令１４のコミット命令がまだ実行されていない状態で新たなコミット命令（命令２１

）をデコードしたので、命令１９～２１の実行キューへの挿入は命令１４の実行まで待たされ、以後の命令のデコードも停止される。

【０１６８】

サイクル９にWステージでは命令１５の結果がグローバルレジスタに出力する結果を退避するためのローカルレジスタに書かれる。以後のサイクルにおいても、特に断らない限りWステージでは同様の処理が行なわれているものとする。

【０１６９】

サイクル１５にロードバッファでｒｅｆｉｌｌが完了し、命令４の結果がローカルレジスタに書かれたとする。サイクル１０～１５の間にはＦステージでの命令フェッチだけが行なわれており、他のパイプラインステージでは処理が行なわれていない。

【０１７０】

サイクル１６にＥステージで命令７が実行され、ロードバッファでは命令１１がロードした結果がローカルレジスタに書かれる。

【０１７１】

サイクル１７にＥステージで命令１０と命令１４が実行される。命令１４は条件に従ってタスクの承認・否認を行なう。条件が成立したと仮定するとタスク１#１、タスク２#１、タスク２#２、タスク３#１が承認され、タスク１#３、タスク１#４、タスク３#３、タスク３#４が否認される。否認されたタスクは実行キューから消去される。承認されたタスクについては、タスクの先頭からグローバルレジスタに出力する命令の直前までの命令が完了済（ローカルレジスタの更新が完了している）であれば消去される。この例では、命令１、命令２、命令４、命令１１、命令１４が消去される。

【０１７２】

一方、命令１４のコミット命令が実行されたため、停止されていた命令デコード部５の処理が再開され、命令１９～２１が実行キューに挿入される。

【０１７３】

サイクル１８にWステージで、命令７の結果をレジスタに出力する。命令７は

既に承認されているので結果はグローバルレジスタに直接書かれて、命令７は実行キューから消去される。

【０１７４】

一方、サイクル１８にＥステージでは命令２１が実行される。命令２１が承認するタスクのうち、タスク３#１を除くものは全て消去されているので、ここではタスク３#１だけが承認され、命令２１は実行キューから消去される。ループ条件が満たされているので他の処理は行なわれない。

【０１７５】

また、サイクル１８にＤステージでは、命令１０～１２が実行キューに挿入される。ループ実行時には並列度が向上するようにタスク番号のリネーミングが行なわれるが、この例では前回のループ時に実行された命令は既に実行キューには存在しないので、タスク番号のリネーミングは行なわれない。

【０１７６】

サイクル１９に命令１０～１２が実行され、サイクル２０に命令１３～１５が実行される。命令１４の条件が満たされなかったとすると、タスク１#３、タスク１#４、タスク２#２、タスク３#３、タスク３#４が承認される。

【０１７７】

サイクル２１にWステージで命令１０によりグローバルレジスタ６番が更新される。Ｅステージでは命令１６が実行される。

【０１７８】

サイクル２２に命令１６がストアバッファに登録される。命令１６は既に承認されているので、ストアバッファエントリのｃｏｍｍｉｔビットは１にセットされる。

【０１７９】

一方、サイクル２２にWステージでは命令１５によりグローバルレジスタ５番が０にセットされる。Ｅステージでは命令２１が実行され、タスク３#１が承認される。ここで、ループ条件が満たされなかったとすると、命令キュー、および承認・否認されていない実行キュー中のタスクは消去される。そして、アドレス生成部２は命令２２を指すように更新される。

【０１８０】

サイクル２３に、ストアバッファに登録された命令１６によりメモリの更新が行なわれる。一方、Ｆステージでは命令２２がフェッチされる。

【０１８１】

サイクル２４に命令２２がデコードされ、サイクル２５に命令２２が実行される。命令２２によりタスク１#１（命令２２）が承認され、アドレス生成部２はグローバルレジスタ３１番で示されるアドレスを指すように更新される。

【０１８２】

図１８の例において、従来のプロセッサで命令１０を命令１４より前に配置する場合には、命令１０が誤って更新したレジスタを元の値に戻すための補正コードが必要になる。よって、命令１０の投機的実行が誤りだった場合には大きなペナルティが生じる。

【０１８３】

一方、本実施形態では補正コードを発行することなく、命令１０を命令１４より前に配置できる。命令の投機的な移動が補正コードなしに行なえるために、広範囲での命令のｏｕｔ－ｏｆ－ｏｒｄｅｒ実行が単純なハードウェアで実現できる。

【０１８４】

また、本実施形態では、ループ構造の前にある命令列をループの内部に含めることも可能である（図１８では行っていない）。このような移動により、ループの投機的実行が従来の方式より広い範囲で可能となる。また、ＶＬＩＷ形式の実現においては、命令の密度が向上するという効果もある。

【０１８５】

次に、本実施形態で実行可能なプログラムコードを生成する、コンパイラのコード生成方式について説明する。

【０１８６】

本発明のコンパイラは、本実施形態で実行可能なプログラムコードを生成するために、従来のコンパイラが行なう処理に加えて、１．命令のタスク割当及びタスクスケジューリング、２．コミット命令を用いた依存関係の表現、３．タスク

間通信を考慮したレジスタ割当および生成・消費フラグの付加といった処理を行なう。以下、これらの処理について説明する。

【０１８７】

まず、命令のタスク割当及びタスクスケジューリングについて説明する。

【０１８８】

一つのタスクウィンドウに存在できるタスクの最大数は、本実施形態のプロセッサが備える実行キューの数となる。本発明のコンパイラは、サブルーチン単位で各命令がこの条件を満たすよう個々のタスクに割り当てる。

【０１８９】

命令のタスク割当においては、まず命令間のデータ依存関係を求めて各命令を複数のデータ依存列に分割する。データ依存関係の解析は、従来のコンパイル手法と同様にして行なえる。

【０１９０】

データ依存列への分割において、図１９のようにデータ依存関係に分岐が存在する場合には、いずれか一方の命令をデータ依存関係のある命令（図１９では命令１）と同じデータ依存列に割り当て、もう一方の命令は別のデータ依存列に割り当てる。また、図２０のようにデータ依存関係に合流が存在する場合には、合流先の命令（図２０では命令３）を、データ依存関係にあるいずれか一方のデータ依存列に含める。

【０１９１】

次に、データ依存列の中から投機的に実行するデータ依存列を選んで、そのデータ依存列を投機的に前方へ移動する。

【０１９２】

データ依存列の投機的移動においては、以下の条件を満たす必要がある。

１．投機的に移動するデータ依存列の長さは実行キューのサイズ以下である

２．プロセッサの任意の状態において、投機的に実行している（つまり、コミット命令により承認・否認されていない）データ依存列の最大数は、予め定められた数（実行キューの数より小さい任意数）以下である

３．条件分岐命令間の順序関係は変わらない

４．投機的に移動するデータ依存列に含まれる命令Ａが依存する結果が条件により異なる場合、命令Ａの実行時点で命令Ａがどの結果を利用すべきは確定している（命令Ａが依存する全ての命令は承認・否認されている）

前記１．の条件を満たすよう、投機的に移動したいデータ依存列の長さが実行キューのサイズを超える場合には、該データ依存列を長さが実行キューのサイズより小さい複数のデータ依存列に分割する。

【０１９３】

ループに含まれない命令に関しては、サブルーチンの先頭から該命令に到達し得る制御パスの種類は有限である。よって、このような命令の実行している状態に関しては、投機的なデータ依存列が最大となる制御パスで前記２．が保証されるように命令を移動する。また、ループを構成する基本ブロック内には命令を投機的に移動しないという制約を課すことで、ループに含まれる命令を実行している状態においても前記２．は保証される。

【０１９４】

条件分岐命令を含むデータ依存列を、別の条件分岐命令を越えて移動したい場合には、該データ依存列を条件分岐命令が含まれる部分と含まれない部分に分割し、条件分岐命令を含まない部分のみを移動することで前記３．を保証する。同様に、前記４．を満たさない位置にデータ依存列を移動したい場合には、そのデータ依存列のうち前記４．を満たす部分のデータ依存列だけを移動する。

【０１９５】

最後に、各基本ブロックに存在するデータ依存列をタスクに割り当てる。投機的なデータ依存列はそれぞれ異なるタスクに割り当て、残されたタスクに投機的でないデータ依存列を割り当てる。一つのタスクに含まれる命令数が実行キューのサイズを超える場合（投機的でないデータ依存列を割り当てたタスクに限られる）には、実行キューが一杯になるまでに無条件コミット命令（条件なしでタスクを承認する命令）が実行されるよう、タスクの命令列に無条件コミット命令を挿入する。

【０１９６】

以上が本コンパイラによる命令のタスク割当及びタスクスケジューリングに関

する説明である。なお、投機的でないタスクを投機的なタスクより優先させる実施形態においては、タスクが投機的か否かを判別できるように、予め投機的なタスクのタスク番号を定めておき、コンパイラは投機的なデータ依存列は前記タスク番号のタスクに割り当てるようにする。

【０１９７】

次に、コミット命令を用いた依存関係の表現方法について説明する。

【０１９８】

本実施形態におけるコミット命令と、従来のプロセッサが備える命令の間には以下のような対応関係がある。本コンパイラは、この対応関係に従って、従来の命令をコミット命令に変換することで依存関係を表現する。

【０１９９】

一般的な条件分岐命令にはｂｃｍｔ命令が対応する。ｂｃｍｔ命令が分岐する基本ブロックは変換前の条件分岐命令が分岐する基本ブロックと等しいが、分岐先の基本ブロックに存在する命令の一部は前方へ投機的に移動されている場合もある。

【０２００】

ｂｃｍｔ命令が承認するタスク群には、そのｂｃｍｔ命令の実行時点では投機的でない未承認タスク群、及び否認するタスク群を指定する。また、ｂｃｍｔ命令が否認するタスク群としては、そのｂｃｍｔ命令の実行時点で未否認である投機的なタスク群のうち、該ｂｃｍｔ命令の条件が成立した際には到達し得ない制御パスに存在していた命令が割り当てられたタスク群を指定する。

【０２０１】

タスクスケジューリングが完了した時点で、分岐条件が満たされない場合にのみ実行される命令群が全て前記条件分岐命令より前に配置されている場合には、前記条件分岐命令はｃｍｔ命令に対応する。ｃｍｔ命令が承認・否認するタスク群の指定はｂｃｍｔ命令の場合と同様である。

【０２０２】

ループの繰り返しのために使用される条件分岐命令にはｌｃｍｔ命令が対応する。ｌｃｍｔ命令の分岐先は変換前の条件分岐命令の分岐先と等しい。ｌｃｍｔ

命令の１回目の実行時点では投機的でない未承認タスク群のうち、ループを構成しないタスク群を１回目のループ実行時にのみ承認するタスクに指定し、残りのタスク群を、ループを構成するタスクとして指定する。

【０２０３】

サブルーチンコール命令にはｊｃｍｔ命令が対応し、サブルーチンリターン命令にはｒｃｍｔ命令が対応する。ｊｃｍｔ命令、ｒｃｍｔ命令が承認するタスクとしてはｂｃｍｔ命令と同様の指定を行なう。

【０２０４】

従来のプロセッサにおけるｃｈｅｃｋ命令（“Ｍｅｍｏｒｙ　Ｃｏｎｆｌｉｃｔ　Ｂｕｆｆｅｒ　ｆｏｒ　Ａｃｈｉｅｖｉｎｇ　Ｍｅｍｏｒｙ　Ｄｉｓａｍｂｉｇｕａｔｉｏｎ　ｉｎ　Ｃｏｍｐｉｌｅ－Ｔｉｍｅ　Ｃｏｄｅ　Ｓｃｈｅｄｕｌｅ”、米国特許５６９４５７７）に対応するのがｃｃｍｔ命令である。

【０２０５】

ｄｃｍｔ命令はｖａｌｕｅ　ｐｒｅｄｉｃｔｉｏｎを行なう命令を含むタスクの最後に挿入し、予測を行なった命令の出力レジスタと該タスクを入力オペランドとして指定する。

【０２０６】

最後にタスク間通信を考慮したレジスタ割当及び生成・消費フラグの付加について説明する。

【０２０７】

タスクスケジューリングとコミット命令の発行が行なわれた後で、タスクウィンドウ単位で命令スケジューリングが行なわれる。命令スケジューリングにおいては、同タスクに属する命令の順序関係は変えてはならないが、他のタスクとの間の順序関係は自由に変えてよい。

【０２０８】

異なるタスクに属する命令間にデータ依存関係が存在する場合には、その配置順序に従って命令に生成・消費フラグを付加する。データを生成する命令が前に配置されている場合には、その命令の出力オペランドに生成フラグが付加される。一方、データを消費する命令が先に配置されている場合には、その命令の入力

オペランドに消費フラグが付加される。

【０２０９】

次に、スケジュールされた命令に対して、レジスタ割当を行なう。

【０２１０】

本実施形態において、グローバルレジスタを使用する必要があるのは、１．ある命令が入力として用いる値を生成するタスクが複数存在する場合、２．あるタスクに属する命令の演算結果を複数のローカルレジスタセットに書く必要がある場合、３．サブルーチン間のインタフェースにレジスタを使用する必要がある場合である。グローバルレジスタの割当については、従来のコンパイラの手法が適用できる。

【０２１１】

上記の場合以外では、本コンパイラはローカルレジスタを割り当てる。ローカルレジスタの割当は、各ローカルレジスタを読み出せるタスク群ごとに行なうことを除けば、基本的には従来のコンパイラと同様の手法が適用できる。ただし、ローカルレジスタ割当の対象とならないタスクの命令によりローカルレジスタが更新される場合には、該ローカルレジスタを入力とする命令の入力オペランドに対するレジスタ割当が行なわれることになる。

【０２１２】

また、ローカルレジスタの生存区間は、そのローカルレジスタを更新する命令が属するタスクウィンドウの先頭から、そのローカルレジスタを使用する命令が属するタスクウィンドウの、次のタスクウィンドウの終端までと考える。

【０２１３】

本実施形態におけるレジスタ表現の一例としては、ローカルレジスタとグローバルレジスタに対して異なるレジスタ番号を用いる表現法が考えられる。異なるローカルレジスタセットに対してはレジスタ番号を重複して用い、結果をローカルレジスタに出力する命令に関しては、更新するローカルレジスタセットの番号を付加して表現する。

【０２１４】

さて、以下では、図２１の記号レジスタで記述されたプログラムコードに対し

て、本方式に基づきコード生成を行なった際の動作について説明する。この例はタスクの実行キューへの割当をコンパイラで行なう実施形態を対象としているため、図２１のプログラムコードは命令の配置アドレスにより、その命令が使用する演算ユニットが決まるＶＬＩＷ方式で表現されている。

【０２１５】

この例では、実施形態として命令実行ユニットが３、実行キューが各命令実行ユニットごとに４つずつの計１２、各実行ユニットのサイズは８としている。

【０２１６】

まず、本発明のコンパイラは、与えられたコードをデータ依存列に分割する。図２１のプログラムコードは、図２２のように１１のデータ依存列に分割される。

【０２１７】

次に、投機的に移動するデータ依存列を選択して移動し、全体のコード長が最短になるようにスケジュールする。この例では、データ依存列４，５，７，８，１０が選択されたものとする。その結果、図２３のようなプログラムコードが得られる。

【０２１８】

最後に、各データ依存列にタスク番号を割り当てると図２４のようになる。図２４では命令が使用する演算ユニットは該命令のアドレスで決定し、命令ニーモニックの前に付加された番号１～４により演算ユニットで処理されるタスクを区別することで、タスクウィンドウごとに１２種類のタスクを表現する。

【０２１９】

以上が命令のタスク割当及びタスクスケジューリングの例である。次に、命令間の依存関係を表現するため、特定の命令をコミット命令に変換する。

【０２２０】

図２４では、条件分岐命令（ｂｅｑ，ｂｎｅ）及びサブルーチンから復帰するためのレジスタ間接ジャンプ命令（ｊｒ）がコミット命令に変換する対象となる。

【０２２１】

図２４の最初に表れるｂｎｅ命令の分岐条件が満たされない場合にのみ実行される命令群は、全てこのｂｎｅ命令の以前に配置されている。よって、最初のｂｎｅ命令はｃｍｔ命令に変換される。他のｂｎｅ命令、ｂｅｑ命令はｂｃｍｔ命令に変換され、ｊｒ命令はｒｃｍｔ命令に変換される。

【０２２２】

図２５がコミット命令発行後のプログラムコードである。コミット命令が承認・否認するタスク群については、タスクが属する命令実行ユニットの番号と、命令実行ユニットごとのタスク番号（図２４で命令ニーモニックに付加された番号）の組で表現している。

【０２２３】

図２４のプログラムコードにレジスタ割当及び生成・消費フラグを付加することで、本実施形態で動作するプログラムコードが得られる。

【０２２４】

この例では、記号レジスタ＄１１２と＄１１３を更新する命令がそれぞれ２つ存在するため、これらの記号レジスタにはグローバルレジスタを割り当てる。サブルーチン間のインタフェースに利用されるレジスタに対してもグローバルレジスタが使用される。他のレジスタについてはローカルレジスタを割り当てる。

【０２２５】

最後に、異なる実行キューに属する命令間にデータ依存関係がある場合には、データの受け渡しに利用されるレジスタに生成フラグ（Ｐ：）または消費フラグ（Ｃ：）を付加する。以上のようにして得られた最終的なプログラムコードを図２５に示す。

【０２２６】

図２５のプログラムコードでは、“#”のついたレジスタがローカルレジスタを示し、他のレジスタはグローバルレジスタを示す。また、“#”の前に番号のついたローカルレジスタを出力とする命令は、その番号でしめされるローカルレジスタセットの、指定されたローカルレジスタを更新する。

【０２２７】

なお、本実施形態におけるコンパイラはソフトウェアとしても実現可能である

。また、本実施形態におけるコンパイラは、コンピュータに所定の手段を実行させるための（あるいはコンピュータを所定の手段として機能させるための、あるいはコンピュータに所定の機能を実現させるための）プログラムを記録したコンピュータ読取り可能な記録媒体としても実施できる。

【０２２８】

本発明は、上述した実施の形態に限定されるものではなく、その技術的範囲において種々変形して実施することができる。

【０２２９】

【発明の効果】

本発明によれば、命令間の依存関係をコミット命令及びレジスタの生成・消費フラグを用いて表現するようにしたので、より少ないハードウェア量で従来のプロセッサよりも広範囲でのｏｕｔ－ｏｆ－ｏｒｄｅｒ命令実行が可能になり、かつ命令が投機的か否かをコンパイラで明示し、投機的でない命令を投機的な命令より優先して実行することで、不必要な命令が演算器を占有することによる性能低下を回避し、プロセッサの性能を向上させることが可能となる。

【図面の簡単な説明】

【図１】　本発明の第１の実施形態に係るプロセッサの構成例を示す図。

【図２】　本発明の第１の実施形態におけるパイプライン処理の一例を示すフローチャート。

【図３】　演算器割当部をタスクと実行キューの関係を保存するマップにより実現する場合について説明するための図。

【図４】　実行キューの構成例を示す図。

【図５】　オペランド状態判定部が持つレジスタスコアボードの一例を示す図。

【図６】　ロード・ストア命令処理の一例を示すフローチャート。

【図７】　ストアバッファの構成例を示す図。

【図８】　ロードバッファの構成例を示す図。

【図９】　ロード命令をロードバッファに登録するための処理の一例を示すフローチャート。

【図１０】　コンフリクトバッファの構成例を示す図。

【図１１】　ｃｍｔ命令を含まないプログラムの一例を示す図。

【図１２】　ｃｍｔ命令を含むプログラムの一例を示す図。

【図１３】　ｂｃｍｔ命令を含まないプログラムの一例を示す図。

【図１４】　ｂｃｍｔ命令を含むプログラムの一例を示す図。

【図１５】　ｌｃｍｔ命令を含まないプログラムの一例を示す図。

【図１６】　ｌｃｍｔ命令を含むプログラムの一例を示す図。

【図１７】　ｄｃｍｔ命令を含むプログラムの一例を示す図。

【図１８】　プログラムの一例を示す図。

【図１９】　データ依存列の分岐について説明するための図。

【図２０】　データ依存列の合流について説明するための図。

【図２１】　記号レジスタで記述されたプログラムの一例を示す図。

【図２２】　図２１のプログラムに存在するデータ依存列を説明するための図。

【図２３】　図２１のプログラムにデータ依存列の投機的移動を施した結果の一例を示す図。

【図２４】　図２３のプログラムにタスク割当を施した結果の一例を示す図。

【図２５】　図２４のプログラムにコミット命令への変換を施した結果の一例を示す図。

【図２６】　図２５のプログラムにレジスタ割当及び生成・消費フラグの不可を施した結果の一例を示す図。

【図２７】　ｐｒｅｄｉｃａｔｅ付き命令を含まないプログラムの一例を示す図。

【図２８】　ｐｒｅｄｉｃａｔｅ付き命令を含むプログラムの一例を示す図。

【符号の説明】

１…メモリ

２…アドレス生成部

３…命令フェッチ部

４…命令キュー

５…命令デコード部

６…タスクウィンドウ識別子生成部

７…演算器割当部

８～１０…実行キュー

１１…オペランド状態判定部

１２～１４…命令実行ユニット

１５～１７…ローカルレジスタ

１８…グローバルレジスタ

１９…グローバルレジスタ更新部

２０…ロードバッファ

２１…ストアバッファ

２２…アドレスバス

２３…命令フェッチバス

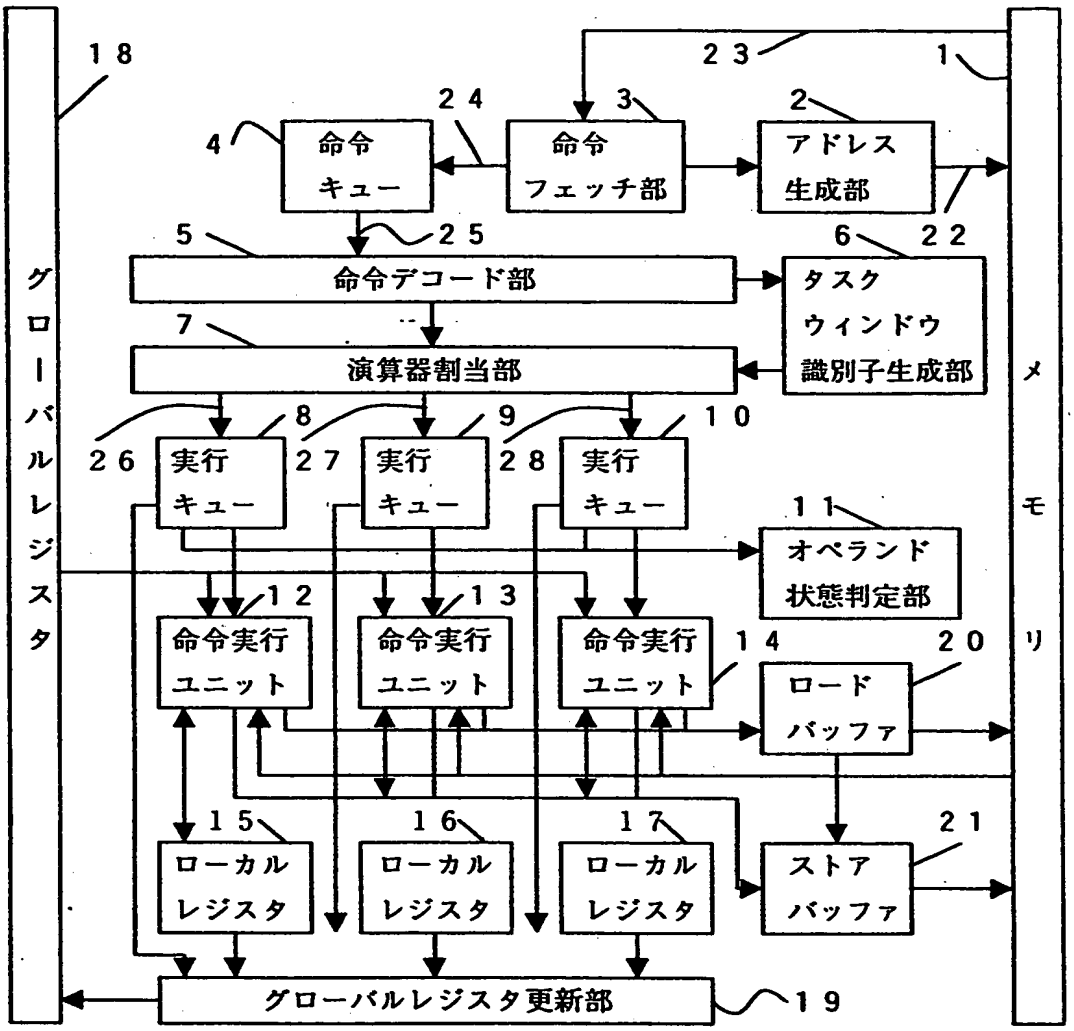
２４…命令キューバス

２５…デコードバス

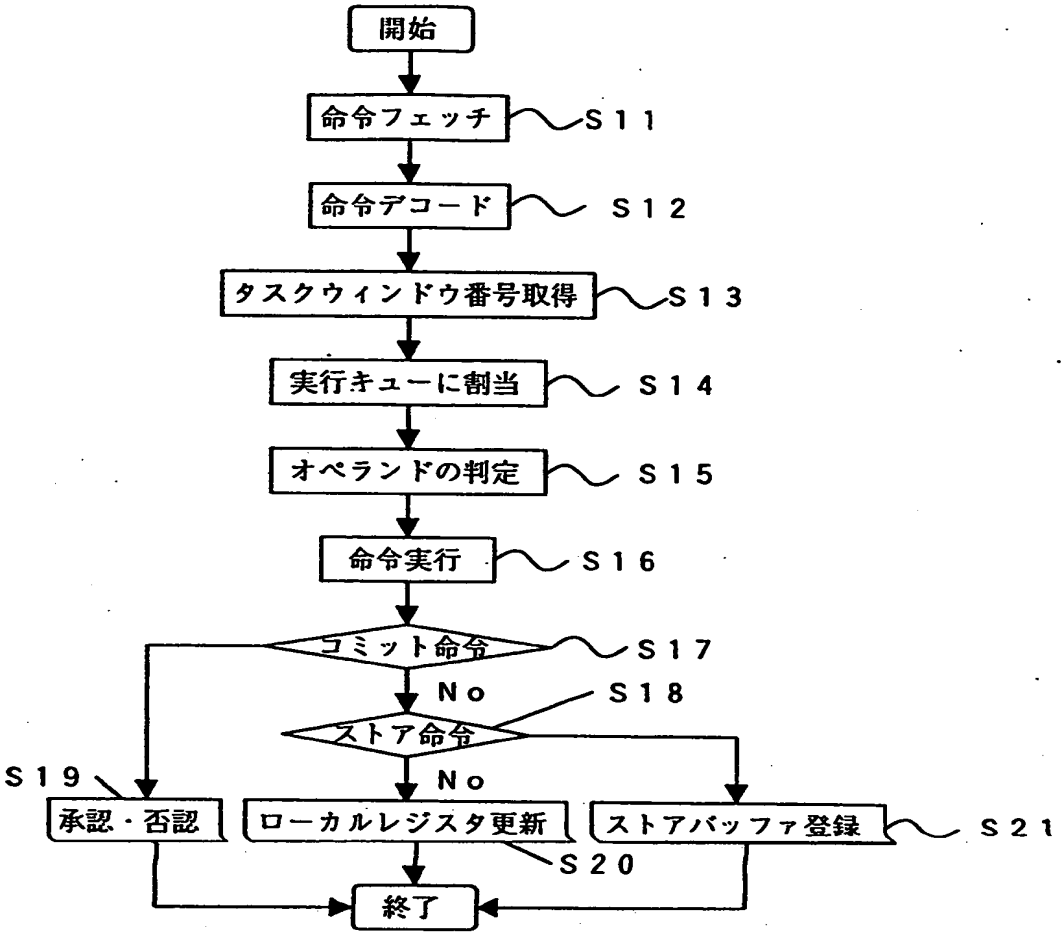
２６～２８…実行キューバス

【書類名】　図面

【図１】


１８
２３
１
２４
３
２
４
命令
キュー
命令
フェッチ部
アドレス
生成部
５
２５
６
２２
命令デコード部
タスク
ウィンドウ
識別子生成部
７
演算器割当部
グローバルレジスタ
メモリ
８
９
１０
２６
実行
キュー
２７
実行
キュー
２８
実行
キュー
１１
オペランド
状態判定部
１２
１３
命令実行
ユニット
命令実行
ユニット
命令実行
ユニット
１４
２０
ロード
バッファ
１５
１６
１７
２１
ローカル
レジスタ
ローカル
レジスタ
ローカル
レジスタ
ストア
バッファ
グローバルレジスタ更新部
１９

【図２】



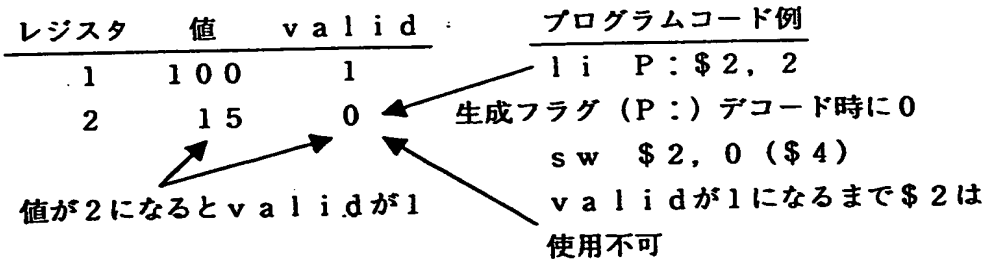
【図３】

タスク	実行キュー	有効
1	5	0
2	3	1
3	1	1
4	4	1

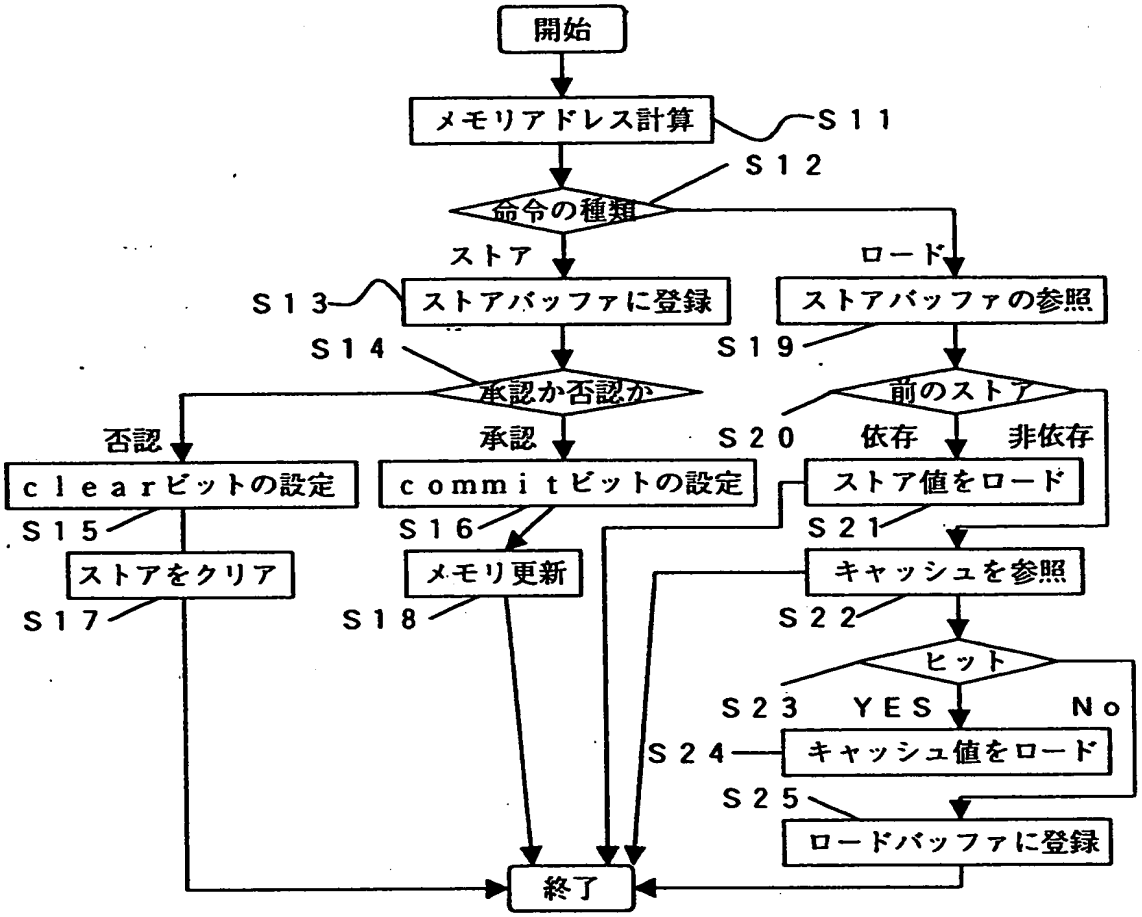
【図４】

命令の種類	オペランド情報	例外情報	
ｌｗ	7，3，0（0）		←ｔａｉｌ
ｓｌｌ	3，4，0（6）		←ｅｘｅｃｕｔｅ
ａｄｄｉｕ	4，5，0（１００）		←ｃｏｍｍｉｔ
ａｄｄｕ	5，6，0（0）		←ｈｅａｄ

【図５】



【図６】



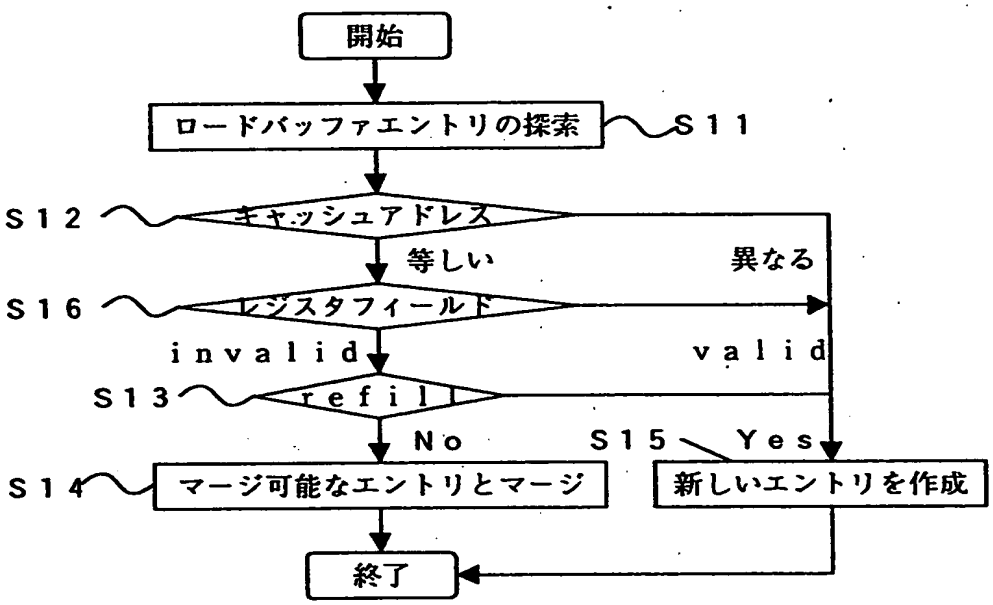
【図７】

アドレス	ストア値	ストア幅	TWID	CMT	CLR	
10046e3c	3d70	H	1	0	0	←tail
10002a1e	1f	B	0	0	1	
10015d60	1eb94	W	0	1	0	←head

【図８】

アドレス	レジスタ	valid	RFL	
100143e0	02,08,15,22,10,34,01,05	00101001	0	←tail
10003160	03,05,14,13,07,04,05,08	01000101	1	←head
	‥			

【図９】



【図１０】

タスク	ロードアドレス	ロード幅	CNF	VLD
3	1000923a	H	0	1
4	1001024c	W	1	0
2	1002149d	B	1	1
	‥			

【図１１】

```
        lbu.e   $3, Ch_1_Glob($0)
        bne     $3, $6, $L48
        lw.e    $5, Int_Glob($0)
        addiu   $7, $7, -1
        move    $2, $0
        subu    $3, $7, $5
        sw --   $3, 0($4)
$L48:
        addiu   $6, $0, 5
```

【図１２】

```
1:lbu.e     $3, Ch_1_Glob($0)
2:lw.e      $5, Int_Glob($0)
2:addiu     $7, $7, -1
3:move      $2, $0
2:subu      $3, $7, $5
2:sw        $3, 0($4)
1:cmt.ne -- $3, $6,
             |1, 2, 3|, |2, 3|
1:addiu     $6, $0, 5
```

【図１３】

```
        lw.e    $4, Int_Glob($0)
        slti    $3, $4, 101
        bne     $3, $0, $L14
        j       $L9
        sw --   $0, 0($56)
$L14:   addiu   $3, $0, 3
```

【図１４】

```
        1:lw.e      $4, Int_Glob($0)
        2:sw        $0, 0($56)
        1:slti      $3, $4, 101
        1:bcmt.ne   $3, $0, $L14,
                     |1, 2|, |2|
        1:j         $L9
$L14:   2:addiu     $3, $0, 3
```

【図１５】

```
        lw      $3, 0 ($57)
        sw      $16, 16 ($sp)
$L41:   lw      $9, 0 ($11)
        addiu   $11, $11, 16
        sw      $9, 0 ($3)
        bne     $11, $10, $L41
```

【図１６】

```
1:lw          $3, 0 ($57)
3:lw          $9:P, 0 ($11)
3:addiu       $11, $11, 16
4:sw          $9, 0 ($3)
2:sw          $16, 16 ($sp)
3:lcmt.ne     $11, $10, $L41,
               |3, 4|, |1, 2|
```

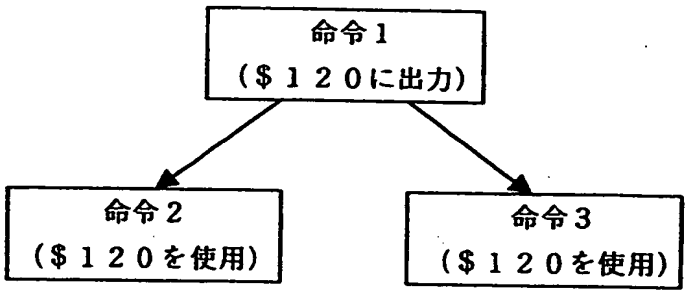
【図１７】

```
1:lw          $3, 0 ($57)
1:addu        $9, $3, $4
3:d.addu      $9, $3, $4
3:sw          $9, 0 ($57)
3:dcmt--      $9, |3|
```

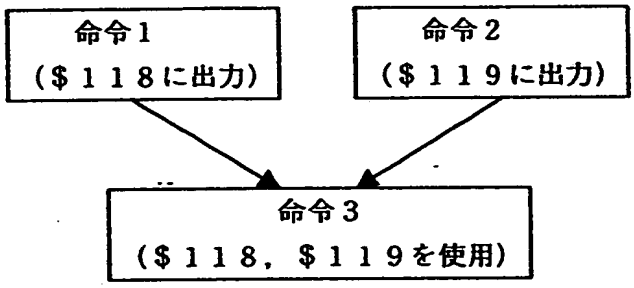
【図１８】

```
        1:move  P:$_5, $4                       [1]
        1:addiu  P:$_1, $0, 65                  [2]
        1:addiu  P:$5, $0, 1                    [3]
        1:lw  $_1, 0 ($_5)                      [4]
        1:nop                                   [5]
        1:nop                                   [6]
        1:addiu  P:$6, $_1, 10                  [7]
        1:nop                                   [8]
        1:nop                                   [9]
  $L46:
        3:addiu  $6, $6, -1                     [10]
        2:lbu.e  $_2, Ch1_Glob ($0)             [11]
        3:lw.e  P:$1_3, Int_Glob ($0)           [12]
        3:subu  P:$_4, $6, $_3                  [13]
        2:bcmt.ne  $_2, $_1, $L48               [14]
           |1_1, 1_3, 2_1, 2_2, 3_1, 3_3, 3_4|, |1_3, 1_4,
3_3, 3_4|
        4:move  P:$5, $0                        [15]
        4:sw  $_4, 0 ($_5)                      [16]
        2:nop                                   [17]
        4:nop                                   [18]
$L48:
        3:nop                                   [19]
        2:nop                                   [20]
        1:lcmt.ne  $5, $0, $L46                 [21]
           |1_3, 1_4, 2_2, 3_1, 3_3, 3_4|, ||
        1:rcmt  $31                             [22]
           |1_1|
        1:nop                                   [23]
        1:nop                                   [24]
```

【図１９】


命令１
（＄１２０に出力）
命令２
（＄１２０を使用）
命令３
（＄１２０を使用）

【図２０】


命令１
（＄１１８に出力）
命令２
（＄１１９に出力）
命令３
（＄１１８，＄１１９を使用）

【図２１】

```
reflect:
        move $114, $6; move $113, $5; move $112, $4
        slti $115, $114, 2; nop; nop
        bne $115, $0, $L109; nop; nop
        lw.e $117, boardsize($0); addiu $116, $113, 1; nop
        subu $113, $117, $116; nop; nop
$L109:
        addiu $118, $0, 1; nop; nop
        beq $114, $118, $L111; nop; nop
        addiu $119, $0, 3; nop; nop
        bne $114, $119, $L110; nop; nop
$L111:
        lw.e $121, boardsize($0); addiu $120, $112, 1; nop
        subu $112, $121, $120; nop; nop
$L110:
        lw.e $123, boardsize($0); nop; nop
        mult $113, $123; nop; nop
        nop; nop; nop
        nop; nop; nop
        mflo $124; nop; nop
        jr $31; addu $2, $124, $112; nop
```

【図２２】

```
(基本ブロック１)
 データ依存列１    move  $１１４, $６
                   slti  $１１５, $１１４, ２
                   bne   $１１５, $０, $Ｌ１０９
 データ依存列２    move  $１１３, $５
 データ依存列３    move  $１１２, $４

(基本ブロック２)
 データ依存列４    lw. e $１１７, boardsize ($０)
 データ依存列５    addiu $１１６, $１１３, １
                   subu  $１１３, $１１７, $１１６

(基本ブロック３)
 データ依存列６    addiu $１１８, $０, １
                   beq   $１１４, $１１８, $Ｌ１１１

(基本ブロック４)
 データ依存列７    addiu $１１９, $０, ３
                   bne   $１１４, $１１９, $Ｌ１１０

(基本ブロック５)
 データ依存列８    lw. e $１２１, boardsize ($０)
 データ依存列９    addiu $１２０, $１１２, １
                   subu  $１１２, $１２１, $１２０

(基本ブロック６)
 データ依存列１０  lw. e $１２３, boardsize ($０)
                   mult  $１１３, $１２３
                   nop
                   nop
                   mflo  $１２４
                   addu  $２, $１２４, $１１２
 データ依存列１１  jr    $３１
```

【図２３】

```
reflect:
        lw.e $123,boardsize($0); move $114,$6; move $11
3,$5;
        mult $113,$123; slti $115,$114,2; move $112,$4
        nop; lw.e $117,boardsize($0); addiu $116,$113,1
        nop; bne $115,$0,$L109; subu $113,$117,$116
$L109:
        mflo $124; addiu $118,$0,1; addiu $119,$0,3
        nop; beq $114,$118,$L111; nop
        nop; bne $114,$119,$L110; nop
$L111:
        addiu $120,$112,1; lw.e $121,boardsize($0); nop
        subu $112,$121,$120; nop; nop
$L110:
        addiu $2,$124,$112; jr $31; nop
```

【図２４】

```
reflect:
 3:lw.e $123,boardsize($0); 1:move $114,$6; 1:move $11
3,$5;
 3:mult $113,$123; 1:slti $115,$114,2; 2:move $112,$4
 3:nop; 3:lw.e $117,boardsize($0); 3:addiu $116,$113,1
 3:nop; 1:bne $115,$0,$L109; 3:subu $113,$117,$116
SL109:
 3:mflo $124; 1:addiu $118,$0,1; 3:addiu $119,$0,3
 3:nop; 1:beq $114,$118,$L111; 3:nop
 1:nop; 1:bne $114,$119,$L110; 1:nop
SL111:
 1:addiu $120,$112,1; 1:lw.e $121,boardsize($0); 1:nop
 1:subu $112,$121,$120; 1:nop; 1:nop
SL110:
 1:addiu $2,$124,$112; 1:jr $31; 1:nop
```

【図２５】

```
reflect:
 3:lw.e $123,boardsize($0); 1:move $114,$6; 1:move $1:
3,$5:
 3:mult $113,$123; 1:slti $115,$114,2; 2:move $112,$4
 3:nop; 3:lw.e $117,boardsize($0); 3:addiu $116,$113,1
 3:nop; 1:cmt.ne $115,$0,; 3:subu $113,$117,$116
                |2_1,2_3,3_1,3_2,3_3|,|2_3,3_3|
 3:mflo $124; 1:addiu $118,$0,1; 3:addiu $119,$0,3
 3:nop; 1:bcmt.eq $114,$118,$L111,; 3:nop
                |1_3,2_1,3_3,3_4|,|3_3,3_4|
 1:nop; 1:bcmt.ne $114,$119,$L110,; 1:nop
                |1_1,2_1,3_1|,||
$L111:
 1:addiu $120,$112,1; 1:lw.e $121,boardsize($0); 1:nop
 1:subu $112,$121,$120; 1:nop; 1:nop
$L110:
 1:addiu $2,$124,$112; 1:rcmt $31,; 1:nop
                                |1_1,2_1|
```

【図２６】

```
reflect:
 3:lw.e $_1,boardsize($0); 1:move P:$_1,$6; 1:move P:$3,
$5:
 3:mult $3,$_1; 1:slti $_1,$_1,2; 2:move P:$7,$4
 3:nop; 3:lw.e P:$3_2,boardsize($0); 3:addiu $_1,$3,1
 3:nop; 1:cmt.ne $_1,$0,; 3:subu $3,$_2,$_1
          |2_1,2_3,3_1,3_2,3_3|,|2_3,3_3|
 3:mflo P:$_2; 1:addiu $_2,$0,1; 3:addiu P:$2_3,$0,3
 3:nop; 1:bcmt.eq $_1,$_2,$L111,; 3:nop
          |1_3,2_1,3_3,3_4|,|3_3,3_4|
 1:nop; 1:bcmt.ne $_1,$_3,$L110,; 1:nop
          |1_1,2_1,3_1|,||
SL111:
 1:addiu $_3,$7,1; 1:lw.e P:$1_4,boardsize($0); 1:nop
 1:subu $7,$_4,$_3; 1:nop; 1:nop
SL110:
 1:addiu $2,$_2,$7; 1:rcmt $31,; 1:nop
                              |1_1,2_1|
```

【図２７】

```
      beq  r3, r4, L2
      li   r5, 0
      j    L1
L2:   sll  r6, r10, 2
      li   r5, 1
L1:   move r2, r5
```

【図２８】

(1) pseq <p1, p2>, r3, r4
(2) <p2> li r5, 0
(3) <p1> sll r6, r10, 2
(4) <p1> li r5, 1
(5) move r2, r5

【書類名】　要約書

【要約】

【課題】　広い範囲でのｏｕｔ－ｏｆ－ｏｒｄｅｒ実行と、不必要な命令実行が生じた場合の性能低下の度合を小さくすることを可能とする中央演算装置の提供。

【解決手段】　プログラムが予め定められたデータ依存性を有する命令列の単位で区切られ、各命令列間の制御依存性はコミット命令により表現され、データ依存性はフラグを命令が使用するレジスタに付与することにより表現され、識別番号を命令毎に付与する識別番号付与手段６と、命令を演算実行ユニット１２～１４に属する複数のバッファ８～１０に割り当てる演算器割当手段７と、特定の命令列が前記コミット命令の実行により承認された場合に、レジスタを更新させるレジスタ更新手段１９と、メモリを更新させるメモリ更新手段２１とを備えたことを特徴とする中央演算装置。

【選択図】　図１

認定・付加情報

特許出願の番号	平成１１年　特許願　第２６７８８９号
受付番号	５９９００９１９４３９
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１１年　９月２７日

＜認定情報・付加情報＞

【提出日】　平成11年 9月22日

次頁無

出　願　人　履　歴　情　報

識別番号　　〔０００００３０７８〕

１．変更年月日　　１９９０年　８月２２日
［変更理由］　新規登録
住　所　神奈川県川崎市幸区堀川町７２番地
氏　名　株式会社東芝